VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MECHANISMUS PRO UPGRADE BIOSU V LINUXU

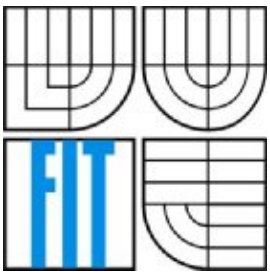
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IGOR MARIŠČÁK

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MECHANISMUS PRO UPGRADE BIOSU V LINUXU

GENERIC BIOS UPDATE MECHANISM FOR LINUX

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. IGOR MARIŠČÁK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ KAŠPÁREK

BRNO 2008

Vysoké učení technické v Brně - Fakulta informačních technologií

Centrum výpočetní techniky Akademický rok 2007/2008

Zadání diplomové práce

Řešitel: **Mariščák Igor, Bc.**
Obor: Informační systémy
Téma: **Mechanismus pro upgrade BIOSu v Linuxu**
Kategorie: Operační systémy

Pokyny:

1. Definujte potřebné funkce a SW rozhraní pro ovladač flash paměti.
2. Vytvořte ovladač flash paměti pro alespoň jeden FWH (např. SST ST49FL004B), který bude používat stejné rozhraní jako stávající MTD ovladače, bude modulární a zaměnitelný s jinými FWH ovladaši beze změny ostatního SW. Požadované funkce jsou: smazání sektoru-čipu, čtení a zápis bajtu/sektoru/čipu a správa ochrany sektorů dle možností FWH.
3. Vytvořte HPM Daemon který bude tvořit rozhraní mezi navrženým ovladačem a OpenIMPI prostředím.
4. Vytvořte nástroje pro otestování ovladače i daemona a vlastní sadu s testy

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVR-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Kašpárek Tomáš, Ing.**, CVT FIT VUT
Datum zadání: 24. září 2007
Datum odevzdání: 19. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Ing. Jaroslav Zendulka, CSc.
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Bc. Igor Mariščák**
Id studenta: 49277
Bytem: Záhradnická 238, 059 91 Veľký Slavkov
Narozen: 07. 10. 1982, Poprad
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Mechanismus pro upgrade BIOSu v Linuxu
Vedoucí/školitel VŠKP: Kašpárek Tomáš, Ing.
Ústav: Centrum výpočetní techniky
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	..... Igor Mavčič
Nabyvatel	Autor

Abstrakt

Táto práca popisuje vytvorenie jednoduchého ovládača pre flash pamäť BIOSu spôsobom prístupu do fyzickej pamäti počítača. Aj keď je BIOS jednou zo základných systémových komponent, neexistuje pre neho štandardizovaný mechanizmus pre aktualizáciu. Účelom práce je vytvoriť modul ovládača využitím existujúceho rozhrania subsystému MTD, navrhnúť a implementovať ovládač pre jedno špecifické zariadenie do jadra operačného systému Linuxu. Ďalej približuje metódu povolenia zápisu do registrov flash pamäti použitím konfiguračného súboru.

Kľúčové slová

MTD, Memory Technology Device, ovládač, Linux (jadro 2.6), programovanie modulu jadra, FWH, firmware, BIOS upgrade, flash pamäť, upgrade mechanizmus, mapovanie I/O pamäti.

Abstract

This work provides overview of creating of a simple driver for the BIOS flash memory by accessing the physical computer memory. Although, the BIOS is one of a system's core components, there is no standardized update mechanism approach. Purpose of thesis is to create module driver by taking advantage of existing interface subsystem MTD, to suggest and implement driver for one specific device to Linux kernel operating system. Also explains technique allowing write access to registers of the flash memory with utilization of configuration file.

Keywords

MTD, Memory Technology Device, driver, Linux (kernel 2.6), module kernel programming, FWH, firmware, BIOS update, flash memory, update mechanism, remap I/O memory.

Citace

Mariščák Igor: Mechanismus pro upgrade BIOSu v Linuxu. Brno, 2008, diplomová práce, FIT VUT v Brně.

Mechanismus pro upgrade BIOSu v Linuxu

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Tomáše Kašpárka.
Ďalšie pomocné informácie mi poskytol vedúci týmu Thomas Kastner z firmy ADVANTECH Europe GmbH, s ktorou som úzko spolupracoval.
Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Igor Mariščák
19.5.2008

Poďakovanie

Na tomto mieste by som chcel poďakovať vedúcemu práce za ochotu pomôcť riešiť problémy pri konzultáciách.
Ďalej rád by som poďakoval svojmu spolubývajúcemu Lacovi za cenné rady a pripomienky; hlavne svojím rodičom i celej rodine, ktorí ma podporovali v štúdiu na vysokej škole, ale aj všetkým kamarátom za ich nápady a veselú náladu, ktorou ma obdarovali.

Obsah

1 Úvod

1.1 Motivácia a cieľ

Základné dosky mávajú uložený BIOS v prídavnej pamäti, kde je možné prepísať - upgradovať obsah novšou verziou. Pre vykonanie upgrade BIOSu môže byť veľa dôvodov. Napríklad ak matičná doska neprijíma, nedetekuje nový vysokokapacitný disk, alebo ak BIOS nespolupracuje s určitým špecifickým hardwarom. Potom aktualizovanie BIOSu novou verziou môže odstrániť tieto problémy.

V dnešnej dobe upgrade BIOSu alebo nahrávanie nového firmware do zariadenia nie je ničím výnimočným a s tým počítajú už samotný výrobcovia dosiek. V niektorých prípadoch už priamo samotná doska podporuje vykonanie upgrade použitím zabudovaného softwaru ešte pred spusteným ostatných súčastí systému (možné zistiť v manuály dosky). Vtedy sa jednoducho skopíruje obsah z diskety do flash pamäti čipu. Je potrebné sa uistiť, že súbor s binárnymi dátami, nový BIOS, obsiahnutý na diskete je presne určený pre značku a typ použitej matičnej dosky (odporúčaním je sťahovať nový BIOS priamo od výrobcu).

Ak však doska nepodporuje upgrade priamo, naskytá sa možnosť vytvoriť spúšťaciu disketu, ktorá sama po reštarte počítača nabútuje a zavedie nahrávací program. Pripraviť takúto disketu je triviálne. Stačí stiahnuť s internetu predpripravený program od konkrétneho výrobcu a zavádzaciu disketu vytvoriť podľa pokynov. Najdôležitejšou podmienkou je mať funkčnú disketu a potrebnú mechaniku. V súčasnosti, pri zastaranom užívaní disketovej mechaniky, je možnosťou vytvoriť rovnakým postupom bútovací CD/DVD kompaktný disk. Pri takomto postupe je úspešnosť upgradu BIOSu takmer 100%.

Situácia sa však komplikuje ak používaným operačným systémom je Linux, kde výrobca vždy nedodáva jednoduchú aplikáciu pre upgrade na vytvorenie takejto diskety. Dostupný zadarmo býva len samotný binárny kód BIOSu. Taktiež je nutné, aby bola nejaká osoba fyzicky prítomná pri počítači, aby založila do mechaniky médium s novým obsahom BIOSu. Pri správe vzdialených serverov alebo telekomunikačných zariadení nedostupnosťou fyzického prístupu je najjednoduchšie prevádzať upgrade BIOSu prostredníctvom internetu alebo vyhradeného komunikačného kanála.

Podobne ako iné operačné systémy aj Linux je súhrnné označenie skupiny programov, ktoré zabezpečujú komunikáciu užívateľa s periférnymi zariadeniami, starajú sa o efektívne využitie operačnej pamäte a pridelenie času procesora, organizujú súborové údaje na pamäťových médiách, v neposlednom rade aj poskytujú zabezpečenie proti neoprávnenej manipulácií a komunikáciu s inými počítačmi v sieti.

Hlavným cieľom práce je navrhnúť a vytvoriť mechanizmus pre upgrade BIOSu v operačnom systéme Linux, aby bolo možné jednoduchým spôsobom nahrávať nový obsah do flash pamäti, zaručením pre čo najviac možných zariadení zmenou konfiguračného súboru.

Táto práca je pokračovaním Semestrálneho projektu, ktorej súčasťou bolo naštudovať a pripraviť prostriedky pre upgrade BIOSu. Týmito prostriedkami sú obslužné programami prístupu k technickému vybaveniu počítača, čiže ovládače komunikujúce s daným typom zariadenia. Diplomová práca nadväzuje na kapitoly dva a tri, ktoré jednoduchým návodom približujú ako vytvárať ovládače pre zariadenia v Linuxe využitím technológie subsystému MTD. Pre správne pochopenie čo musí zahrňovať taký ovládač bolo nutné sa oboznámiť ako funguje linuxové jadro.

1.2 Štruktúra práce

Druhá kapitola rozoberá problematiku ako komunikuje výkonná časť, jadro operačného systému so zariadením prostredníctvom rozhraní jednotlivých logických vrstiev operačného systému a definuje rozdelenie zariadení na typy podľa určitých charakteristických vlastností.

Kapitola tri prináša pohľad na existujúci subsystém MTD a obsahuje stručný návod ako napísať ovládač pre typ zariadenia akým je flash pamäť.

Celá štvrtá kapitola je venovaná tvorbe ovládača. Obsahuje podrobný postup akým je možné inicializovať zariadenie a registrovať jeho vlastnosti. Dôležitou úlohou je správne pochopiť nízkoúrovňový prístup k pamäti flash, fyzické realizovanie komunikácie a povolenie zápisu. Ďalej pojednáva o prideľovaní pamäti v jadre a mapovaní fyzickej pamäti pre získanie prístupu. V poslednom rade opisuje priebeh kompilácie ovládača a nový spôsob prípravy dávkového súboru prekladu pre zostavovací systém jadra Linuxu (verzia 2.6).

V piatej kapitole je popísaný postup povolenia zápisu do registrov flash pamäti vytvorenou aplikáciou využitím konfiguračného súboru. Záverečná kapitola zobrazuje len návrh prepojenia na systém správy.

2 Linuxové ovládače zariadení

Ovládač slúži pre vytvorenie jednotného prístupu pre zariadenia rôzneho typu. Môže sa jednať o ovládač klávesnice, CD-ROM, SCSI, IDE, RAM disku a pod. Zariadenie môže byť prakticky akákoľvek časť počítača. Môže to byť interné alebo periférne (externé) zariadenie. Taktiež nemusí byť len fyzické, ale môže to byť napr. špeciálny súbor, ktorý exportuje nejakú funkciu aplikáciám a nie je pripojený priamo na hardware. Dal by sa tiež označiť ako softwarové zariadenie (software device), ktoré zabezpečuje prepojenie medzi aplikáciou a súborovým systémom.

Vytvoriť ovládač nemusí byť náročnejšie ako napísať malý program, ale povedané jednoznačne, prístup k ošetrovaniu neočakávaných náhodných chýb či výnimiek musí byť dôslednejší a rozvážnejší. Pri písaní ovládačov je potrebné poznať niektoré nové knižnice, ktoré sa pri riešení štandardných úkonov vôbec nevyužívajú (nie sú dostupné alebo nemôžu byť použité) a podobne sa riešia odlišné problémy ako pri písaní programu. Je potrebné si uvedomiť súvislosť, že implementovaním modulu (kód ovládača) sa dotvára časť jadra systému a že aplikácie sú ďaleko viac chránené voči chybám. Zlé napísaný modul (či už úmyselne alebo nie) môže fatálne ohroziť stabilitu a funkčnosť celého systému.

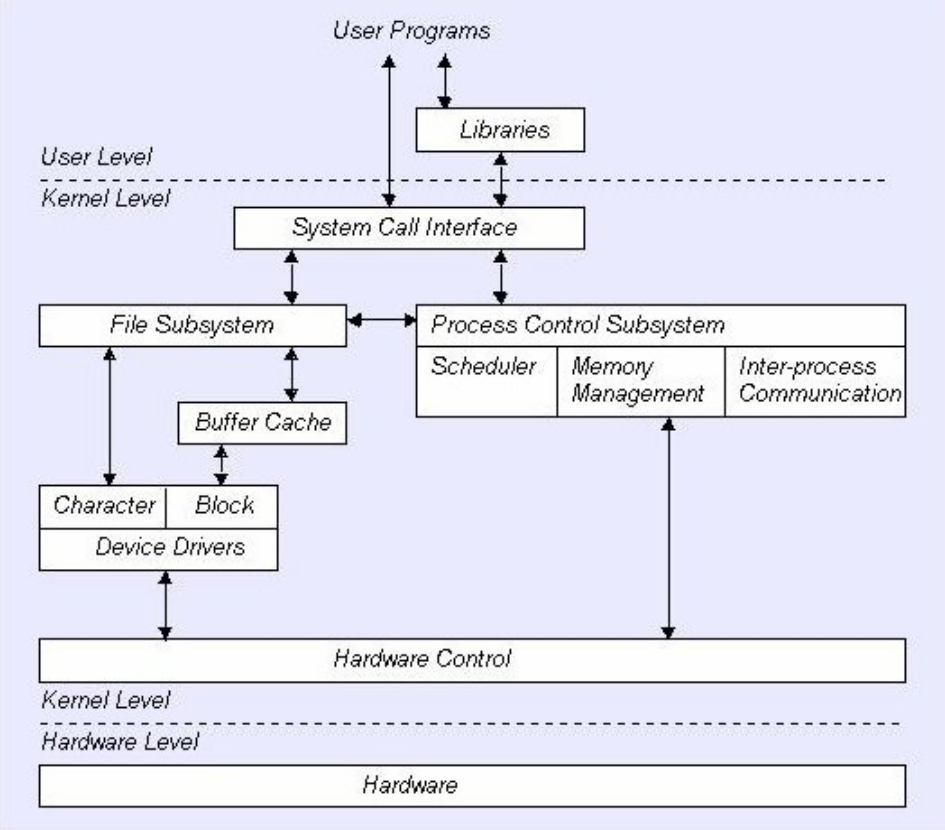
Dôležité je, že modul beží v režime jadra a má právo pristupovať k zariadeniam priamo. Jadro vytvára riadiacu vrstvu, prostredníka medzi užívateľským režimom a hardwarom (logické vrstvy sú zobrazené na obrázku 2.1). Umožňuje užívateľskému programu komunikovať s modulom, pričom ten nemusí ani tušiť s akým reálnym zariadením pracuje.

Niektoré prídavne funkcie jadra môžu byť staticky zakompilované do jadra alebo ponechané ako samostatné moduly. Takéto moduly - ovládače zariadení sa v Linuxe ľahko nahrávajú/vkladajú dynamicky do jadra použitím príkazu **insmod *modul***. Analogicky sa odoberajú/vymazávajú z jadra operačného systému aplikovaním príkazu **rmmod *modul***. Potom nie je žiadny problém pridať napr. podporu pre nový súborový systém alebo nové zariadenie za behu systému. Nie je potrebné prepísať kód jadra, stačí len pripojiť modul. Pre inteligentnejšie zavádzanie alebo odoberanie modulov z jadra slúži program **modprobe *modul***, ktorý používa konfiguračný súbor pre načítanie závislostí medzi modulmi. V systéme existuje hierarchia (možné vygenerovať príkazom **depmod**) určujúca, ktorý modul musí byť zavedený do pamäti pred ostanými. Takže modprobe je komplexnejší nástroj, nadstavba nad insmod. Zoznam zavedených modulov v jadre sa zisťuje príkazom **lsmod**.

2.1 Systémové volania

Procesy bežiace v užívateľskom (neprivilegovanom) priestore nemajú právo komunikovať priamo s hardwarom, majú povolené vykonávať obmedzenú inštrukčnú sadu a v pamäti majú prístup len tam, kam im jadro operačného systému povolí. Preto, ak chce proces pristupovať k nejakému zariadeniu,

musí najprv zavolať jadro, povedať mu čo chce urobiť, a jadro to vykoná za neho. K tomu slúžia systémové volania (**system calls** alebo skrátene len **syscalls**). Systémové volania sú súčasťou jadra a počas behu operačného systému sú uchovávané v tabuľke s adresami jednotlivých volaní.



Obrázok 2.1: Komunikácia z užívateľského priestoru k zariadeniu cez jadro OS[I].

2.1.1 Súborové rozhranie a ovládače zariadení

Súbory sú v Unixe zvlášť dôležité, lebo poskytujú zrozumiteľné a konzistentné rozhranie pre služby operačného systému a zariadení. Programy môžu obecne pracovať s diskovými súbormi, sériovými portami, tlačiarňami a ďalšími zariadeniami rovnakým spôsobom, ako keby pracovali so súborom.

V unixových systémoch je **skutočne skoro všetko** (existujú výnimky, medzi ktoré patrí sieťové spojenie) reprezentované pomocou súborov alebo **sprístupnené pomocou** špeciálnych **súborov**. I keď je možné nájsť jemné rozdiely od konvenčných súborov, obecný princíp zostáva stále rovnaký. Unix obsluhuje vstupne/výstupné operácie so súbormi pomocou určitých systémových volaní, ale existuje tiež celá rada knižničných funkcií, štandardná V/V knižnica (stdio), ktorá prácu zefektívňuje.

[I] Prevzaté zo zdroja: http://i.cmpnet.com/nc/unixworld/graphics/010.fig2.jpg

V podstate je potrebné si zapamätať 5 základných funkcií: open, close, read, write a ioctl. Tieto funkcie tvoria vlastné rozhranie operačného systému. Ovládač zariadenia je súbor s nízko úrovňovým rozhraním pre riadenie systémového hardwaru, ktorým sa zaoberám v mojej práci a vytáram ho.

Súbory umiestnené v adresári zariadení /dev sú používané rovnakým spôsobom (rovnaká funkcia sa použije pri prístupe k normálnemu užívateľskému súboru): dajú sa otvoriť, je možné z nich čítať, zapisovať do nich, a zavrieť. Systémové volanie ioctl poskytuje niektoré nevyhnutné riadenie, špecifické pre daný hardware, takže jeho použitie sa od zariadenia k zariadeniu líši. Z toho dôvodu nemusí byť volanie prenosné z jedného systému na druhý. Okrem toho každý ovládač definuje vlastnú sadu príkazov ioctl.

2.1.2 Nízko-úrovňový prístup k súborom

Každý bežiaci proces má pridružených niekoľko deskriptorov súborov. Sú to čísla používané pre prístup otvoreným súborom alebo zariadeniam. Ich počet závisí od nastavenia operačného systému. Každý spustený program má zakaždým tri z týchto deskriptorov už otvorené (štandardný vstup 0; štandardný výstup 1; štandardný chybový výstup 2).

Ďalšie deskriptory sa získavajú a prácu s nimi zabezpečujú nasledujúce volania (konkrétne definície funkcií sú zahrnuté s ich popisom). Obrázok 2.2 vyznačuje rozhranie a sumarizuje použitie týchto volaní.

- open

*int open (const char *path, int oflags);*

Zabezpečí prepojenie prístupovej cesty k súboru alebo zariadeniu a deskriptora. V prípade úspechu vráti nový deskriptor súboru (vždy sa jedná o najmenšie nepoužité kladné číslo). Je jedinečný a nie je zdieľaný iným bežiacim procesom. Cesta s názvom súboru alebo zariadenia, ktoré sa má otvoriť, je predávaná ako parameter *path* a akcie, ktoré sú možné s otvoreným súborom vykonávať špecifikuje parameter *oflags*. Ten sa definuje ako bitové OR povinného režimu prístupu (otvorenie len pre čítanie O_RDONLY; otvorenie len pre zápis O_WRONLY; otvorenie aj pre čítanie a zápis O_RDWR) a ďalších nepovinných parametrov.

- read

*size_t read (int fildes, const void *buf, size_t nbytes);*

Prečíta n bajtov *nbytes* dát zo súboru združeného s deskriptorom *fildes* a umiestni ich do dátovej oblasti *buf*. Vráti počet skutočne prečítaných dát, ktorý môže byť menší ako je požadované. Ak volanie vráti hodnotu 0, nebolo už čo čítať; bol dosiahnutý koniec súboru.

- write

*size_t write (int fildes, const void *buf, size_t nbytes);*

Volanie zapíše prvých n bajtov z premennej *buf* do súboru združeného deskriptorom *fildes*. Vráti počet skutočne zapísaných bajtov. V prípade chyby deskriptora súboru alebo ak je ovládač zariadenia citlivý na veľkosť bloku, môže ohlásiť menší počet bajtov ako je uvedené v premennej *nbytes*. Ak funkcia vráti hodnotu 0, znamená to, že neboli zapísané žiadne dáta.

- close

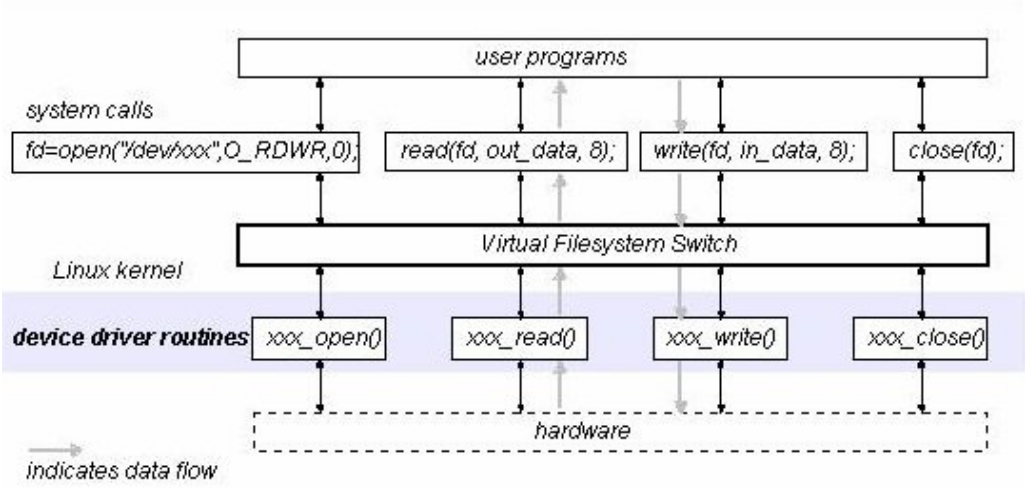
int close (int fildes);

Slúži k prerušeniu spojenia medzi deskriptorom súboru *fildes* a skutočným súborom. Deskriptor je potom možné znovu použiť. V prípade úspešného vykonania operácie vráti hodnotu 0. Testovanie návratovej hodnoty príkazu je dôležité, pretože niektoré súborové systémy nemusia oznámiť chybu zápisu pokiaľ nie je súbor uzavretý.

- ioctl

int ioctl (int fildes, int cmd, ...);

Charakteristické rysy hardwaru sú sprístupnené prostredníctvom tohto volania. Poskytuje rozhranie pre riadenie chovania zariadenia. Vlastné volanie ioctl spúšťa funkciu určenú parametrom *cmd* nad objektom, ktorý popisuje deskriptor *fildes*. V závislosti na funkciách podporovaných konkrétnym zariadeným sa môžu objaviť ďalšie parametre.



Obrázok 2.2: Input/Output interface poskytovaný nezávisle na zariadení[II].

U spomínaných systémových volaní je v prípade výskytu chyby vrátená hodnota -1, pričom sa nastaví globálna premenná errno na hodnotu udávajúcu dôvod zlyhania. Počet súborov, ktoré môže mať každý proces súčasne otvorené, je obmedzený. Toto číslo je definované konštantou OPEN_MAX v hlavičkovom súbore limits.h (štandard POSIX vyžaduje minimálne 16). Existuje mnoho ďalších systémových volaní, ktoré umožňujú programu riadiť spôsob použitia súboru a vracajú informácie o jeho stave. Sú zvyčajne dokumentované v 2. sekcii unixových manuálových stránok.

[II] Prevzaté zo zdroja: http://i.cmpnet.com/nc/unixworld/graphics/010.fig3.jpg

- lseek

off_t lseek (int fildes, off_t offset, int whence);

Nastavuje ukazovateľ zápisu/čítania deskriptoru súboru *fildes*, takže pomocou neho sa určuje, kde bude nasledovať čítanie alebo zápis. Ukazovateľ je možné nastaviť na absolútne miesto v súbore (SEEK_SET) alebo na relatívnu pozíciu vzhľadom k aktuálnej pozícii (SEEK_CUR) alebo konci súboru (SEEK_END). Parameter *whence* udáva spomínaný spôsob akým sa použije *offset*. Funkcia vracia ofset meraný v bajtoch od začiatku súboru, na ktorý je nastavený ukazovateľ. Typ off_t používaný pri vyhľadávacích operáciách je typom závislým na implementácii, je definovaný v hlavičkovom súbore sys/types.h.

2.2 Triedy zariadení a moduly

Ovládače zariadení pozostávajú zo sady funkcií nazývaných rutiny (routines), ktoré kontrolujú periférne zariadenia pripojené ku počítaču. Operačný systém bežne poskytuje jednotné a uniformné rozhranie ku všetkým vonkajším zariadeniam (vystihuje obrázok 2.2). Reprezentácia zariadení na vyššej úrovni abstrakcie dovoľuje vidieť veľké množstvo V/V vstupne/výstupných (Input/Output) operácií ako postupnosť, sekvenciu bajtov.

Každý modul (ovládač zariadenia) je preto zvyčajne implementovaný jedným z týchto typov, zaradený do skupiny ako znakový, blokový alebo sieťový modul. Rozdelenie na takto definované typy, alebo triedy, nie je tak prísne strohé. Je možné vystavať obrovský modul implementovaním rozličných ovládačov v jednom kuse zdrojového kódu. Však dobrý prístup pri programovaní je zvyčajne vytvoriť rozličné moduly pre každú novú funkcionalitu, ktorú implementujú, pretože dekompozícia je kľúčovým elementom ku **škálovateľnosti a rozšíriteľnosti**. Spôsob ako Linux prezentuje a pozerá sa na zariadenia charakterizujú tri základné typy zariadení.

2.3 Rozdelenie typov zariadení

2.3.1 Znakové zariadenia

Znakové zariadenia sú tie, ktoré môžu byť sprístupnené bajt po bajte, podobne ako súbor. Ovládače pre znakové zariadenia sa starajú o implementáciu tohto chovania. Takéto ovládače prinajmenšom zvyčajne definujú operácie systémových volaní: otvor - open, čítaj - read, zapíš - write, kontroluj - ioctl , zavri - close, a iné

Príkladom je textová konzola alebo sériový port a podobné zariadenia reprezentujúce abstrakciu sekvenčného toku dát. Prístup nie je zabezpečený žiadnou vyrovnávacou pamäťou (bufferom). Je len jeden závažný rozdiel medzi znakovými zariadeniami a bežným súborom, a to v tom, že sa v súbore dovoľuje vždy posúvať späť dozadu alebo ďalej dopredu, kým vo väčšine

znakových zariadení, ktoré sú akými si dátovými kanálmi, je prístup iba jednosmerne sekvenčný. K väčšine ovládačov v Linuxe je umožnené sa dostať cez súborový systém. Znakové zariadenia sú reprezentované pomocou uzlov (nodes) a sú uložené v adresári /dev (napr.: /dev/tty1, /dev/lp0, /dev/console, a pod.)

2.3.2 Blokové zariadenia

Podobne aj blokové zariadenia sú prístupné v Linuxe pomocou uzlov súborového systému, ktoré sú uložené v adresári /dev. Blokové zariadenia sú označované niekedy ako tie, ktoré môžu sami hostiť súborový systém akým je pevný disk (/dev/hda0). Vo väčšine Unixových systémov, blokové zariadenia môžu len zachádzať s V/V operáciami prenášajúcimi jeden alebo mnohonásobné celé bloky dát, ktoré majú zvyčajne 512 alebo väčšiu (vždy mocniny dvoch) dĺžku bajtov. Oproti tomu je v Linuxe dovolené aplikáciám čítať a zapisovať do blokových zariadení ako do znakových - povolený prenos hocijakého počtu bajtov na jeden krát. Výsledkom toho je, že sa blokové zariadenia odlišujú od znakových len v spôsobe riadenia vnútorne v jadre (zabezpečený vyrovnávacou pamäťou) a práve preto aj v programovom rozhraní ovládačov. Aj keď majú úplne odlišné rozhranie v jadre a existujú rozdielnosti medzi nimi, sú navonok pre užívateľa transparentné a prehľadné.

Po zadaní príkazu na výpis obsahu adresára:

root@test:/home/igor/fwh_dev# **ls -l /dev** tak je možné vidieť v type súboru

c = character device (znakové zariadenie) alebo **b = block device** (blokové zariadenie).

2.3.3 Sieťové zariadenia

Všetky sieťové transakcie sú konané prostredníctvom rozhraní, čo skoro vždy predstavuje fyzické zariadenie (sieťovú kartu), ktorá vymieňa dáta s inými hostiteľmi v sieti alebo to môže byť prosté softwarové zariadenie, akým je špeciálna slučka (loopback) zasielajúca dáta späto väzobne na vstup rozhrania. Mnohé sieťové spojenia (obzvlášť tie používajúce TCP) sú prúdovo orientované, ale sieťové zariadenia starajú len o odosielanie a prijímanie dátových balíčkov (packets). Potom aj sieťové ovládače nemusia vedieť nič o samotných spojeniach, iba ovládajú pakety. Sieťové rozhrania nie sú tak jednoducho namapované do súborového systému pomocou odpovedajúcich uzlov, ale prístup k nim je prostredníctvom priradenia stále unikátneho mena (napr.: eth0). Komunikácia medzi jadrom a sieťovým ovládačom zariadenia je preto úplne rozdielna od spôsobu ako je použitý pre znakové alebo blokové ovládače.

Existujú aj iné spôsoby klasifikácie zariadení a ovládačov, ktoré sú založené na iných vlastnostiach spomínaných predchádzajúcom texte. Zadaním nie je rozobrať všetky detailne. Vo všeobecnosti všetky ostatné typy ovládačov pracujú s prídavnými vrstvami jadra, doplňujúcimi podporné funkcie špeciálne určené pre konkrétny typ zariadenia.

3 Memory Technology Device (MTD)

Ako prvé je potrebné vysvetliť pojem Memory Technology Device, ktorý sám zachytáva podstatu ovládačov zariadení tohto typu. Voľným prekladom do slovenčiny by mohlo byť spojenie *"technológia pracujúca s pamäťou zariadenia"* ktorá vystihuje prečo som použil tento subsystém.

Cieľom linuxového projektu MTD [4] bolo vytvoriť generický subsystém pre podporu rôznorodých pamäťových zariadení, špeciálne pre druh flash pamäti. Existuje množstvo dostupných flash čipov, ktoré prinášajú početné metódy ako ich preprogramovať. Štruktúra vrstiev MTD oddeľuje komplikovanosť low-level zariadení od high-layer organizácie uložených formátov dát.

Flash pamäť má charakteristiky, ktoré sa nezhodujú úplne s popisom zariadení rozdelených do kategórií na blokové alebo znakové známe v Unixových operačných systémoch. Správanie je podobné blokovým zariadeniam, pretože je pamäť flash organizovaná v sektoroch - blokoch ale má odlišnosti. Napríklad blokové zariadenia nerozlišujú medzi operáciou zápisom dát a čistým vymazaním.

Nasledujúca tabuľka popisuje rozdiely medzi blokovým zariadením a flash pamäťou.

blokové zariadenie [disk]	**MTD zariadenie [flash pamäť]**
ovládač diskových zariadení	MTD ovládač
pozostáva zo sektorov	pozostáva z vymazateľných blokov
sektory sú malé (512, 1024 Byte)	mazateľné bloky sú väčšie (32, 128 KiByte)
podporujú 2 hlavné operácie: čítaj sektor - read(), zapíš sektor - write()	podporujú až tri operácie: čítaj z bloku - read(), zapíš do bloku - write(), vymaž blok - erase()
zlé sektory sú premapované a skryté vo vnútri zariadenia (moderné LBA pevné disky)	poškodené bloky nie sú skryté a mali by byť ošetrené programovo
sektory sa neopotrebúvajú, nemajú túto vlastnosť	mazateľné bloky sa opotrebúvajú (nepoužiteľné poškodené) po približne 10^4 - 10^5cykloch mazania

Tabuľka 1: Základné charakteristiky MTD zariadení.

Zariadenie pre ktoré píšem ovládač je flash pamäť BIOS (skratka z anglického Basic Input/Output System), v ktorom je uložené programové vybavenie osobného počítača (čiže firmware). Vytvára základnú vrstvu abstrakcie nad fyzickými prvkami PC a je mu predané riadenie pri (re)štarte. Prvotnou funkciou BIOSu je identifikovať komponenty na matičnej doske, pevné časti počítača (disky, diskety, CD mechaniky, zvukové a sieťové karty, a pod.), inicializovať interné

zariadenia (generátor hodín, procesor, vyrovnávacie pamäte, I/O kontrolér) a zaručiť, že bude možné natiahnuť ďalšie programy, väčšinou software OS, uložené na týchto médiách. Tento proces prípravy a spustenia operačného systému po zapnutí počítača sa nazýva bútovanie (booting).

Úlohou môjho projektu nie je rozoberať ako prebieha tento proces, ani to ako vytvárať kód uložený v BIOSe, aby správne rozpoznával rozličné zariadenia, ale vytvoriť mechanizmus, spôsob ako ukladať naprogramované binárne dáta do flash pamäte čipu. Dej prepisu nového obsahu BIOSu (upgrade) sa bude uskutočňovať až po ukončenom spustení operačného systému a jeho všetkých služieb.

Pamäť flash slúži ako miesto uloženia programového kódu BIOSu podobne ako integrované pamäti ROM alebo EEPROM na matičnej doske udržuje údaje v pamäti natrvalo (semipermanentne) aj po vypnutí prúdu. Môže byť elektricky vymazaná a znovu preprogramovaná podľa potreby, čo jej dáva výhodu oproti typom pamätí s prístupom len pre čítanie. Na rozdiel od EEPROM je vnútorne organizovaná po blokoch, ktoré sa dajú samostatne programovať bez zmeny obsahu ostatných. Je to technológia, ktorá bola primárne použitá v pamäťových kartách a USB kľúčoch pre ukladanie a presun dát medzi počítačmi. Hneď po inicializácii operačnej pamäti RAM sa typicky BIOS sám skopíruje do RAM pamäti od určitého miesta a ostatné operácie vykonáva z nej kvôli rýchlejšiemu prístupu.

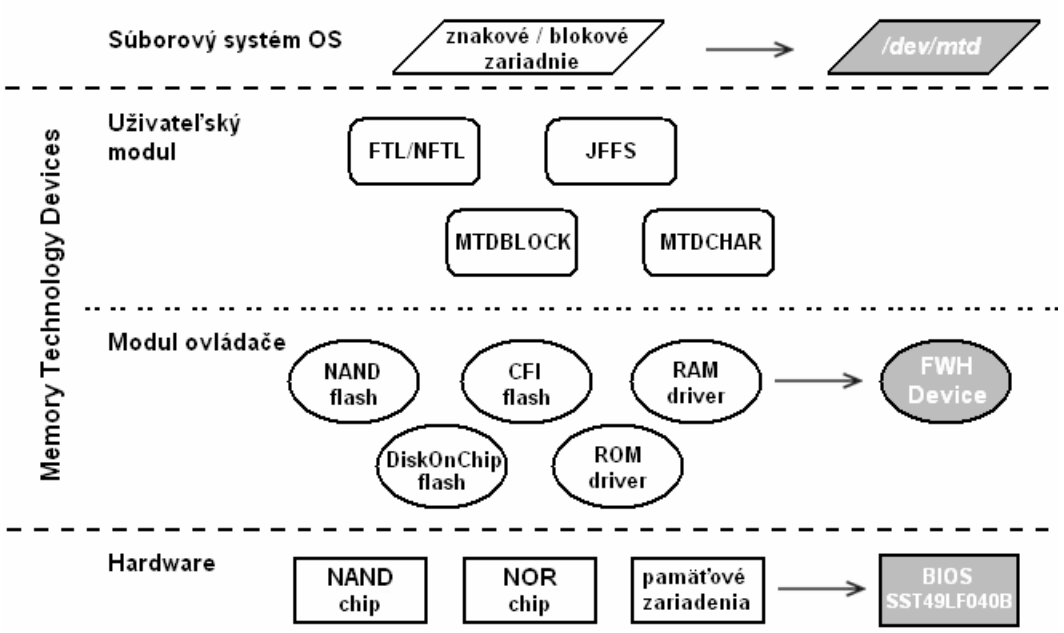
3.1 Náhľad na internú dokumentáciu

Použiteľná dokumentácia k systému MTD (prístupná na stránkach [5]) je podľa autorov už zastarala. Avšak zobrazuje hlavné myšlienky ako postupovať pri tvorení ovládača zariadenia takéhoto subsystému. Ten je vytvorený, aby čo najjednoduchšie umožnil generovať ovládač pre nové technické zariadenie, poskytnutím rozhrania pre ovládač tak aj vyšším vrstvám operačného systému.

Štruktúra MTD systému je zásadne rozdelená na dve samostatné časti, alebo moduly: **užívateľský** a **modul ovládače** (názorné rozdelenie prináša obrázok 3.1).

3.1.1 Rozhranie pre vytváranie ovládačov zariadení

Časť subsystému MTD zaoberajúca sa ovládačmi, ktorá poskytuje jednoduché a základné operácie prístupu k fyzickej pamäti zariadenia: čítaj (read), zapisuj (write) a vymaž (erase), sa nazýva hardvérová vrstva. Táto najnižšia vrstva nemusí vedieť nič o type formátu uložených dát. Poskytuje len možnosti, prostredníctvom ktorých sa kopírujú dáta zvyčajne po jednotlivých bajtoch.



Obrázok 3.1: Detailné rozdelenie vrstiev infraštruktúry MTD.

3.1.2 Modul ovládače

Medzi už vytvorené ovládače zariadenia (prevzaté z oficiálnej stránky projektu [4], sekcia Documentation) patria napríklad:

- pamäte na doskách –na mnohých počítačových doskách nie je schopnosť správne sprístupniť vyrovnávajúcu pamäť niektorých čipov nad určitou hranicou adresného priestoru, a preto operačný systém využije existujúci ovládač MTD ako rozhranie pre lepšiu správu.
- PCMCIA zariadenia – počítačové karty s pamäťami flash využívanými v mobilných zariadeniach sú neustále vyvíjané a zlepšované, sú taktiež podporované MTD ovládačmi.
- Common Flash Interface (CFI) onboard NOR flash – bežný spôsob riešenia, je dobre testovaný a podporovaný s najčastejšie používaným súborovým systémom JFFS2.

Inštrukcie pre napísanie ovládača sú veľmi jednoduché a použil som ich pri tvorbe vlastného ovládača. Stručné vysvetlenie: Najprv je potrebné alokovať miesto v pamäti potrebné na uloženie štruktúry *mtd_info* a potom ju naplniť informáciami o konkrétnom druhu zariadenia. Veľmi potrebné je špecifikovať, v skutočnosti napísať prístupové rutiny k zariadeniu podliehajúce určitým pravidlám a nastaviť ukazovatele štruktúry na tieto funkcie. Posledným bodom je zaregistrovanie (pridanie) ovládača do subsystému pomocou funkcie *add_mtd_device()*. Pri odoberaní modulu z jadra musí sa zavolať opačná funkcia *del_mtd_device()*, ktorá odregistruje ovládač z MTD subsystému.

3.1.3 Užívateľský modul

Užívateľský modul zapuzdruje MTD ovládače, ponúka vyššiu úroveň abstrakcie pre priestor rozhrania operačného systému a prezentuje obsah zariadenia užívateľovi.
V súčasnej dobe sú k dispozícii 4 užívateľské moduly:

- FTL (File Translation Layer)a NFTL (NAND File Translation Layer) - vykonávajú transformáciu nazývanú rovnomerné opotrebovanie. Sektory flash pamäti dokážu vydržať len konečný počet operácií mazania (rádovo 100 000). Vrstva ošetruje opotrebovanie a predlžuje životnosť flash pamäti rovnomernou distribúciou požiadaviek cez všetky sektory. Spoločne FTL a NFTL poskytujú pseudo-blokové zariadenie, na ktorom môže byť umiestnený súborový systém. Niektoré algoritmy použité v týchto moduloch sú patentované a preto obmedzené na zachádzanie.
- JFFS (Journaling Flash File System) - štruktúrovaný súborový systém, ktorý beží priamo nad flash zariadením. Originálne bol napísaný pre pamäti typu NOR, ale s jadrom Linuxu (verzia 2.6) podporuje aj NAND technológiu. Uplatňuje tiež metódu rovnomerného opotrebovania. Zmeny obsahu sú logované v uzloch (metadata spolu s dátami), rozmiestnené po flash pamäti a spojené ukazovateľmi na hlavičky. Platné uzly sa stavajú zastarané, keď je vytvorená ich nová verzia.
- MTDBLOCK - systém, ktorý nevykonáva preklad a poskytuje len rozhranie pre blokové zariadenie priamo nad základným prvkom MTD ovládača. Emuluje pevný disk nad flash pamäťou, skrýva komplikované prístupové procedúry ako je nutnosť pri zápise najprv odpovedajúci sektor vymazať. Nad modulom je možné umiestniť ľubovoľný súborový systém.
- MTDCHAR - prestavuje súvislý pohľad na zariadenia flash pamäti namiesto blokovo orientovaného, priamy prístup na spodnú vrstvu pre občasné ukladanie zmenených informácii.

Vytvoriť užívateľský modul je viac zložitejšie a komplexnejšie. Uvádzam len pre úplnosť a nebudem vytvárať žiadny takýto typ, pretože to nie je súčasťou môjho zadania. Je potrebné napísať dvojicu ohlasovacích funkcií na pridanie a na odoberanie prístupu, ktoré budú volané kedykoľvek, keď sa ovládač pridá do resp. odoberie zo systému. Podobne ako u ovládačov je tiež určené akým spôsobom sa zaregistrujú tieto funkcie v subsystéme MTD.

Zdrojový kód MTD je aktualizovaný v jadre Linuxu (verzia 2.6) a nie je požadovaná žiadna inštalácia alebo stiahnutie prídavných zdrojových a hlavičkových súborov. Ďalšie prípadné informácie sú k dispozícii na stránkach projektu [4] v sekcii Source, kde sa nachádzajú odkazy umožňujúce prezerať zdrojové kódy cez webové rozhranie.

4 Vytvorenie ovládača flash pamäti

Cieľom diplomovej práce je vytvoriť mechanizmus pre upgrade BIOSu, ktorého podstatnou časťou je vytvorenie jednoduchého ovládača v Linuxe využitím spomínanej technológie subsystému MTD. Ten bude slúžiť ako rozhranie pre technické vybavenie (hardware) počítača v procese aktualizácie flash pamäte použitej pre uchovanie BIOSu.

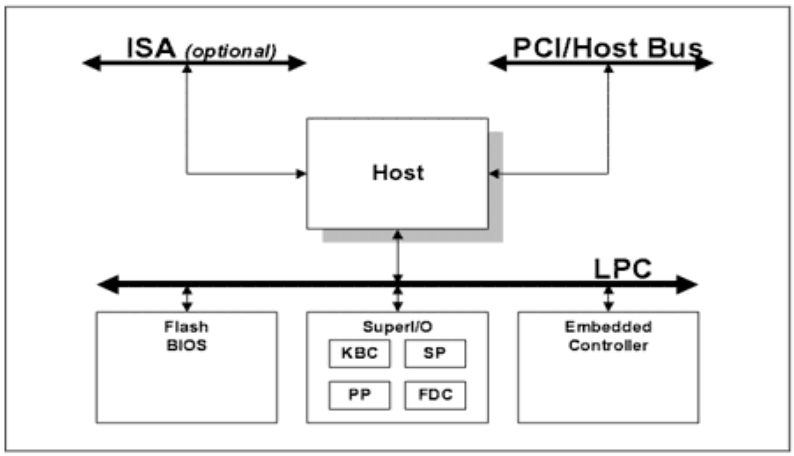
Pri písaní modulu linuxového jadra je veľká výhoda v tom, že sú k dispozícii všetky kódy celého jadra, ktoré je možné prechádzať v stromovej štruktúre (kernel tree), upravovať a znovu používať pri vlastnom tvorení. Dôležitosť zdroja takto získaných informácií, rád a myšlienok je nedoceniteľná.

Vývojovým prostredím je štandardná distribúcia Linuxu s verziou jadra 2.6. Zdrojový kód ovládača by mal byť znovu použiteľný aj s inými distribúciami s minimálnym úsilím potrebných zmien. Na kompiláciu bol použitý štandardný prekladač GCC (verzia 4.1.3) pod operačným systémom Ubuntu 7.10. (prednostne preferovaný) s jadrom Linux (verzia 2.6.22).

Postupne bude v texte detailne vysvetlené ako vytvoriť takýto ovládač (kompletný zdrojový kód je obsiahnutý na priloženom CD) v jazyku ANSI C. Mála poznámka, ďalej spomenuté hlavičkové súbory sa nachádzajú na obvyklom mieste v adresárovej štruktúre zdrojových kódov jadra /usr/src/linux-source.

Pre správne implementovanie ovládača je najskôr potrebné uviesť ako funguje hardwarové zariadenie pamäťový flash prvok. Na uchovanie programu kódu BIOSu sa používali v histórii rozličné pamäte podľa veľkosti dát, spôsobu fyzického skonštruovania. V mojom prípade pracujem s produktom označením **SST49LF040B**, príp. SST49LF004B (rozdiely sú v internom zapojení). Jedná sa o zariadenie flash pamäť, ktoré je navrhnuté na pripojenie k rozhraniu kontroléru matičnej dosky. Ten musí podporovať prepojenie na zbernicu pomenovanú LPC (Low Pin Count) určenú primárne pre prepojenie zariadení s malou šírkou pásma prenosu dát.

Firmware Hub (FWH) je novší štandard kompatibilný s už existujúcim systémom na doske v počítači, ktorý využíva na ukladanie obsahu BIOSu pamäťový flash prvok (chip). Ten však nie je napojený priamo na procesor a dátovú zbernicu, ale namiesto toho je prepojený na spomínaný druh zbernice LPC, ktorá nahradila zbernicu ISA a je súčasťou čipovej sady (chipset) architektúry od firmy Intel (viz. obrázok 4.1). Doplňujúce informácie k špecifikácií LPC ako aj technické parametre čipu sa nachádzajú na konci v prílohe.



Obrázok 4.1: Schéma zapojenia LPC zbernice v systéme[III].

4.1 Zariadenie flash pamäť

Čip SST49LF040B zariadenie je vlastne SuperFlash pamäťové pole pre uloženie dát o veľkosti maximálne 4 Mbit (512 KB) riadené vnútornou kontrolnou logikou s dvoma definovanými módmi rozhraniami.

Sú podporované tieto módy rozhrania:

1. LPC pamäťový mód pre operácie už v zabudovanom systéme počítača kompatibilné so spomínanou Intelovou špecifikáciou rozhrania Low Pin Count Interface Specification.
2. paralelný programový mód určený pre rozhranie s priemyselným štandardom programového vybavenia, ktorým sa ale nebudem ďalej zaoberať.

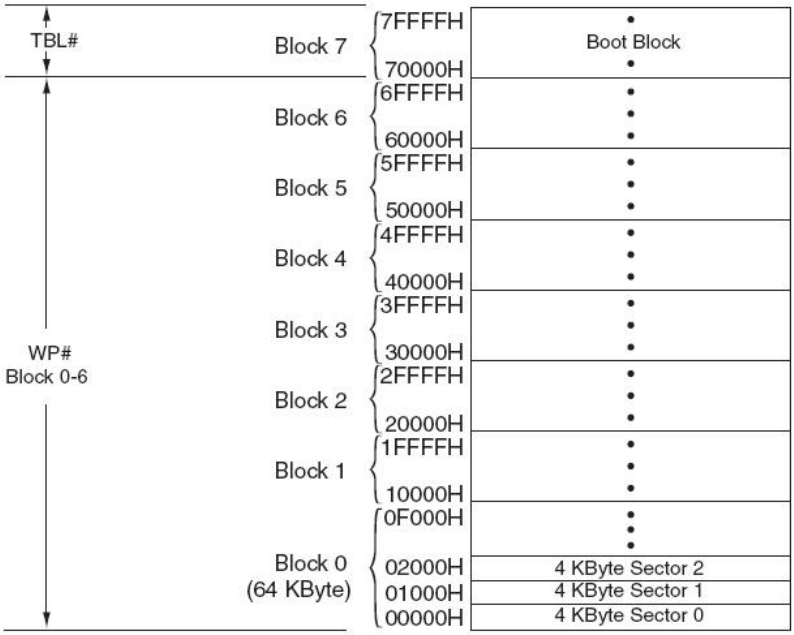
Pamäťový mód LPC poskytuje jednoduchý cyklus čítania/zápisu len jedného bajtu. Celá pamäť čipu môže byť potom vymazaná a preprogramovaná bajt za bajtom pri použití detekcie pomocou hodnôt bitov Toggle Bit a Data# Polling, ktoré indikujú stav dokončenia operácie mazacieho alebo programovacieho cyklu. Táto vlastnosť je veľmi dôležitá a jej funkčnosť bude vysvetlená ďalej. Zabezpečenie proti nechcenému zápisu čip uplatňuje pomocou zabudovanej hardvérovej (konkrétny pin na zariadení) tak aj programovej schémy ochrany dát (Software Data Protection).

4.1.1 Vnútorné oblasti flash pamäti

Ako už bolo spomínané, po spustení počítača je celý obsah flash pamäti namapovaný do operačnej pamäti ako Input/Output pamäť, čo musí byť povolené čipovou sadou na doske. Obsah je však nedostupný z užívateľského priestoru, lebo je nad definovanou hranicou (podrobnosti v odstavci "Registrovanie modulu").

[III] Prevzaté zo zdroja: http://www.svethardware.cz/art_doc-B9F2C0E23681BC0EC12571B1004FF8F5.html

Pre zariadenie flash pamäti platí, že je ich vnútorné členenie rozdelené do blokov. Pre SST49LF040B ale aj pre všetky podobné je zvyčajne táto veľkosť 64 KB. Obrázok 4.2 zobrazuje vnútorné členenie na uniformné sektory (menšie jednotky delenia) a nad nimi prekrývajúce sa bloky. Pre ucelenosť informácií udávam, že najvyšší blok je špeciálne vyčlenený a pomenovaný ako TBL (Top Boot Block), pretože poskytuje prídavnú hardwarovú ochranu proti zápisu pre 16 bútovacích sektorov, t.j. jeden blok. Ostatné bloky sú chránené nezávisle pinom WP (Write Protected). Z hľadiska funkčnosti je správanie úplne rovnaké a súhrne sa označuje ako oblasť *Memory*.



Obrázok 4.2: Rozsah adries oblasti Memory[IV].

Okrem oblasti pre uloženie dát, zahŕňa každý čip aj oblasť pre softwarovú ochranu proti nechcenému zápisu, ktorá je taktiež mapovaná do RAM. Nazýva sa *Register*.

Rozsah adries dekódovaný kontrolérom pre použité zariadenie je vyznačený v tabuľke 3, na základe definícii bitov pre flash pamäť zapojenú ako bútovacie zariadenie. Keďže je možné v systéme používať viac flash čipov pre zvýšenie pamäťovej hustoty, každý z nich musí byť označený ID pinom, a tak je adresovateľný jednoznačne. Pre bútovacie zariadenie (device #0) je prístupový rozsah definovaný ID[3:0] = 1111, pre ďalšie je identifikačné pole sekvenčne dekrementované. Každá štandardná adresa (32 bitov) v definovanom rozsahu operačnej pamäti je preložená ako prístup do oblasti pamäti zariadenia (viz. tabuľka 2). Vytvorený ovládač musí poskytovať užívateľským procesom jednoduchý prístup k obom oblastiam.

[IV] Prevzaté zo špecifikácie zariadenia [7]

A_{31}:A_{24}	A_{23}	A_{22}	A_{21}:A_{19}	A_{18}:A_0
1111 1111	ID[3]	1 = Memory prístup 0 = Register prístup	ID[2:0]	Adresa pamäti zariadenia

Tabuľka 2: Definícia bitov pre adresovanie oblastí pamäti (Device #0).

Označenie zariadenia	Prístup do oblasti	Rozsah adries v systémovej pamäti	Veľkosť pamäti
Device #0 bútovacie	Memory	FFF8 0000H – FFFF FFFFH	512 KB
	Register	FFBC 0000H – FFBF FFFFH	512 KB

Tabuľka 3: Rozsah adries dekódovaný kontrolérom (Device #0).

4.1.2 Registre

Tieto registre sa mapujú na adresy do systémovej pamäti v stanovenej oblasti (špecifikované v tabuľke 3), objavujú sa na príslušných adresách v 4 GB priestore a sú platné pre len pre bútovacie zariadenie Pri čítaní z miesta nepoužitých adries registrov je vždy získaná hodnota 00H. Akýkoľvek prístup čítať alebo zapisovať do registrov počas vnútornej operácie zápisu je ignorovaný.

Pre zariadenie SST49LF040B sú dostupné tri typy registrov:

1. General Purpose Inputs register (všeobecný účel)

   Tento register predáva na výstup aktuálny stav pinov GPI[4:0], ktorými je zariadenie hardvérovo prepojené. Nemá implicitnú hodnotu, pretože len kopíruje hodnotu (slúži len na čítanie). Adresa prístupu k tomuto registru je FFBC0100H pre zariadenie zapojené ako bútovacie, pre iné je namapovaná do miesta v pamäti určeného poradím v systéme.

2. JEDEC ID register

   Poskytuje prístup k informáciám o výrobcovi a identifikácii zariadenia obdŕžané jedným cyklom čítania z definovaných adries. Nasledujúca tabuľka zobrazuje adresy registrov a význam získaných hexadecimálnych hodnôt. Je to pomôckou pri overení správnosti fungovania a prístupu obsahu zariadenia flash pamäti.

Identifikácia	Adresa	Prečítané dáta	Význam
Výrobcu - Manufacturer ID	FFBC 0000H	BFH	SST company
Zariadenia - Device ID	FFBC 0001H	50H	SST49LF040B

Tabuľka 4: Identifikácia produktu SST49LF040B.

3. Block Locking registre (uzamykanie blokov)

   Najdôležitejšie registre ochrany zariadenia SST49LF040B. Poskytujú softwarovú kontrolu pre povolenie zápisu do dátovej oblasti uzamknutím jednotlivých blokov. Aktuálny status konkrétnych registrov je možné čítať ale aj zapisovať nové hodnoty. V každom registri sú význačné dva bity: Write-Lock a Lock-Down.

   - Write-Lock (bit 0) kontroluje stav uzamknutia príslušného bloku. Implicitná hodnota je u všetkých registrov po spustení alebo reštarte zariadenia nastavená na zamknutú, a tým je zabránené meniť obsah dát vykonaním programovacej alebo mazacej operácie do tejto oblasti. Povolením, vynulovaním tohto bitu sa stáva zodpovedajúci blok nechránený. Preto je najprv potrebné pred zápisom vynulovať tento bit, čo je podstatou súčasťou mechanizmu upgradu obsahu čipu rozoberaného v 5. kapitole.
   - Lock-Down (bit 1) kontroluje samotný prístup zápisu do Block Locking registrov. Prednastavený stav tohto bitu je nenastavený čiže odomknutý. Po nastavení bitu je hocijaký pokus zmeniť obsah daného registru ignorovaný. Opätovné vymazanie daného bitu sa prevedie až po resetovaní zariadenia alebo po novom štarte.

   Možné kombinácie hodnôt sú špecifikované v nasledujúcej tabuľke.

Rezervované bity [7:2]	Lock-Down [1]	Write-Lock [0]	Stav uzamknutia	Hexa
0000 00	0	0	Plný prístup	00H
0000 00	0	1	Zápis uzamknutý	01H
0000 00	1	0	Prístup uzamknutý	02H
0000 00	1	1	Uzamknuté všetko	03H

Tabuľka 5: Hodnoty bitov Block Locking registra.

4.1.3 Operácie prístupu

Zadefinovanie vlastností, ktoré má poskytovať daný ovládač je z hľadiska mechanizmu pre upgrade BIOSu zásadná vec. Ovládač poskytuje rozhranie pre kopírovanie dát z užívateľského procesu do zariadenia flash pamäti. Preto musí obsahovať postupy, ako správne pristupovať k tejto fyzickej vrstve.

Každá flash pamäť má výrobcom presne stanovené ako čítať a najmä zapisovať dáta do jej vnútornej pamäti. V prípade SST49LF040B je to dané softwarovou sekvenciou príkazu. Každý zápis bajtu (programovacia operácia) vyžaduje prevedenie série troch bajtov neprerušene v cykle takým spôsobom, že sa postupne vystavuje určitá hodnota na správnu adresu (pamäťový rozsah je daný mapovaním čipu do RAM). Až potom je možné zapísať požadovaný bajt na konkrétnu adresu. Pre vymazanie celého bloku alebo len jedného sektoru (mazacia operácia) je nutné zahrnúť až päť bajtovú

sekvenciu príkazu až za ňou zahrnúť začiatočnú adresu pre vymazanie. Nasledujúca tabuľka predstavuje konkrétne hodnoty programovej schémy prístupu. Čítať dáta je operácia nevyžadujúca žiadne ďalšie sekvencie postupov.

Názov sekvencie príkazu	**Cyklus č.1**		**Cyklus č.2**		**Cyklus č.3**	
	Adresa	Dáta	Adresa	Dáta	Adresa	Dáta
Byte-Program	5555H	AAH	2AAAH	55H	5555H	A0H
Sector-Erase	5555H	AAH	2AAAH	55H	5555H	80H
Block-Erase	5555H	AAH	2AAAH	55H	5555H	80H
Názov sekvencie príkazu	**Cyklus č.4**		**Cyklus č.5**		**Cyklus č.6**	
	Adresa	Dáta	Adresa	Dáta	Adresa	Dáta
Byte-Program	addrBP	dataBP				
Sector-Erase	5555H	AAH	2AAAH	55H	addrSE	30H
Block-Erase	5555H	AAH	2AAAH	55H	addrBE	50H

Tabuľka 6: Softwarová sekvencia príkazu.

4.1.4 Detekcia ukončenia operácii

Zariadenie SST49LF040B poskytuje dva softwarové prostriedky na detekciu ukončenia sekvencie príkazu (programovacieho alebo mazacieho) za účelom optimalizácie systémového času jedného cyklu. Táto programová detekcia zahŕňa dva stavové bity: Data# Polling D[7] a Toggle Bit [D6].

Spôsob detekcie konca zápisu/mazania je spojený s pamäťovým cyklom čítania. Úplné dokončenie neprerušeného cyklu zápisu/mazania je asynchrónne so systémom a preto čítanie stavových bitov môže prebiehať zároveň s dokončovaním tohto cyklu.

- Data# Polling

Ak zariadenie vykonáva vnútornú programovaciu operáciu, akýkoľvek pokus čítať tento dátový bit D[7] produkuje komplement skutočného bitu dát. Po ukončení operácie získaný bit by mal odpovedať skutočnému bitu zapisovaných dát. Hoci ukončenie je signalizované ihneď správnou hodnotou bitu D[7], zvyšné bity výstupu môžu byť stále neplatné a objavia sa na dátovej zbernici až po ďalších úspešných cykloch čítania za určitý interval. Počas vnútornej operácie mazania, snaha čítať bit D[7] produkuje '0', po ukončení hodnotu '1'. Podmienkou získania správneho stavu Data# Polling je adresa v správnom rozsahu.

- Toggle Bit

Počas vnútornej operácie programovania alebo mazania zariadenia, hocijaký nepretržitý pokus čítať bit D[6] produkuje striedavo nuly a jednotky, prepína medzi '0' a '1'. Keď zariadenie dokončí

operáciu, prepínanie sa zastaví. Podobne ako pri Data# Polling signalizovaní ukončenia, ostatné bity výstupu môžu byť stále neplatné a objavia sa až po časovom intervale. Vlastná detekcia stavu neposkytuje hodnoty Toggle Bit, ak je adresa v nesprávnom rozsahu.

Spôsob detekcie môže náhodne niekedy dávať chybný výsledok, platné dáta sa môžu objaviť v rozpore D[7] či D[6]. Aby sa predišlo falošnému zamietnutiu zápisu pri vzniku mylného zistenia, softwarová rutina by mala obsahovať cyklus čítania prístupového miesta dodatočne dvakrát. Ak sú obidve prečítané hodnoty platné ako zapísané dáta, zariadenie ukončilo zapisovací cyklus úspešne, v opačnom prípade nastala vnútorná chyba a hlási zamietnutie pokračovania.

4.2 Návrh modulu ovládača

Dobrým odporúčaním pri písaní zdrojového textu nových častí je používať vhodne zvolený menný priestor funkcií, premenných i makier, a tak pracovať len v oblasti vlastných definícii. Taktiež, aby nedošlo k predefinovaniu globálnych premenných alebo funkcií som explicitne vždy používal pamäťovú triedu static. Kľúčové slovo je uvedené pred každou definíciou. Má to ten význam, že sú identifikátory viditeľné len v rámci súboru a nikde inde. Výhodou je opäť skutočnosť, že nie je potrebné sa obávať kolízií rovnako zvolených mien identifikátorov v ostatných moduloch. V mojom prípade som všetky funkcie, symbolické konštanty a makrá pre zvýšenie čitateľnosti pomenoval s predponou fwh_dev_ príp. FWH_DEV_ (skratka od Firmware Hub Device).

4.2.1 Vstupný bod modulu

Podobne ako bežný program, ktorý musí obsahovať práve jednu hlavnú funkciu pomenovanú main (syntakticky definovaná ako normálna funkcia; odlišuje sa jedine tým, že sa volá prvotne po spustení programu), je potrebné pri každom module ovládača definovať dve funkcie. Prvá sa zavolá v okamihu, keď je modul nahrávaný do jadra a druhá tesne pred jeho odstránením z jadra pri dynamickom vkladaní. Názvy týchto funkcií nie sú striktne určené, užívateľ si ich môže definovať sám. Nutné je však vždy použiť špeciálne makrá jadra, aby bolo možné identifikovať rolu týchto funkcií. Predávaným parametrom makier sú názvy jednotlivých funkcií.

Keď je ovládač zariadenia vkladaný do jadra, musia byť vykonané niektoré prípravné úlohy, ktoré zabezpečia rezervovanie systémových prostriedkov (pamäte RAM) pre fungovanie modulu, nastavenie interných dátových štruktúr, resetovanie fyzického zariadenia na inicializačné hodnoty, atď. Tieto úkony je nutné urobiť ešte pred prvým spustením príslušného zariadenia. V mojom prípade sa o to postará vytvorená funkcia fwh_dev_init(). Pre vypnutie zariadenia a o uvoľnenie všetkých zdrojov, ktoré ovládač alokoval sa použije funkcia fwh_dev_exit(). Deklarácie týchto makier sú v systémovom hlavičkovom súbore linux/init.h

4.2.2 Makrá na zadefinovanie vlastností

Existujú ďalšie špeciálne makrá s parametrami, aby nastavili charakteristické vlastnosti modulu ovládača. Takým je aj makro MODULE_LICENCE na určenie licencie modulu, pod ktorou môže byť použitý, čo ovplyvňuje, ktoré symboly (funkcie, premenné) môžu byť dostupné jadru. Bez tejto deklarácie by sa jadro sťažovalo, vypísalo hlásenie, či dokonca nenatiahlo modul. Keďže použitým parametrom je "*GPL*" príp. "*GPLv2*" spadá pod licenciu GNU (General Public Licence) a dovoľuje každému modifikovať, distribuovať a dokonca predávať produkt so zdrojovým kódom so zaručením rovnakých práv.

Medzi makrá, ktoré pridávajú informácie do modulu patria napríklad MODULE_AUTHOR, MODULE_DESCRIPTION, MODULE_VERSION, ktoré ako ich pomenovanie napovedá určujú meno autora, stručný popis modulu ovládača a použitú verziu. Je úplne jedno, na ktorom mieste v zdrojovom texte modulu sa definujú s príslušnými parametrami, však zaužívaným pravidlom je umiestniť ich pre poriadok dohromady na koniec súboru. Deklarácie týchto makier sú v systémovom hlavičkovom súbore linux/module.h.

Všetky informácie sú po skompilovaní modulu ovládača dostupné po zadaní príkazu **modinfo** a názvu preloženého objektu modulu. Získaný vzorový výpis informácií z výstupu tohto programu je uvedený v prílohe.

Makro __init spôsobí, že funkcia bude vyradená a pamäť, ktorú zaberala, bude uvoľnená ihneď po ukončenej inicializácii. To platí len pre staticky zabudované moduly v jadre pri štartovaní operačného systému, kedy jadro používa funkcie vo fázy inicializácii len raz.

4.2.3 Inicializácia modulu ovládača

O inicializáciu sa stará funkcia fw_dev_init(). V tejto funkcii sa zavolá mnou vytvorená iná funkcia fwh_dev_reg() pre obe spomínané oblasti pamäti flash pomenované Memory a Register. Pre každú oblasť je potrebné alokovať miesto v pamäti pre štruktúru mdt_info a ďalej naplniť túto štruktúru konkrétnymi informáciami. Pre subsystém MTD a jeho ovládače zariadení bola vytvorená mtd_info štruktúra a práca s ňou je veľmi jednoduchá pomocou globálne definovaného ukazovateľa. Definícia tejto štruktúry je uvedená v hlavičkovom súbore linux/mtd/mtd.h.

Pre správne fungovania modulu je potrebné správne určiť aké hodnoty majú byť uložené v mtd_info. Vytvorený ovládač bude mať prístup k systémovej pamäti počítača od určitej adresy start, kde je namapovaný obsah BIOSu. Druhým parametrom je veľkosť pamäti size, s ktorou môže pracovať.

Pri výpisoch vzniknutých problémov potrebuje jadro svoju vlastnú funkciu, pretože beží samostatne bez pomoci štandardných knižníc jazyka C. Ekvivalentom štandardnej funkcie printf() je printk(), ktorá však neumožňuje tlač čísel s pohyblivou čiarkou kvôli fyzickej realizácii prepínaniu kontextu na procesore. Povinným predávaným parametrom je reťazec, ktorý nastavuje prioritu

tlačenej správy. Možné hodnoty definujú makrá v hlavičkovom súbore linux/kernel.h Príkladom je KERN_ALERT, ktorý špecifiku v module druhú najvyššiu prioritu, pretože správa so štandardne malou prioritou sa nemusí zobraziť. V moduloch je možné volať printk(), lebo po zavedení bežia taktiež na úrovni jadra a majú prístup k symbolom

Použitie preddefinovaných úrovní vytvára zdrojový text čitateľnejším. Nastavením úrovne protokolovania sa určí a riadi, ktoré správy budú v skutočnosti zobrazené na konzole alebo zapísané do protokolu syslog. Správy sa väčšinou nezobrazia hneď po vytlačení. Sú totiž uložené vo vyrovnávacej pamäti jadra, ktorá je pre tento druh správ vyhradená. Však čo je práve v nej uložené sa ľahko zistí pomocou programu **dmesg**, ktorý vypíše správy v poradí ako prichádzali od prihlásenia.

4.2.4 Získavanie a uvoľňovanie pamäti

Pamäť je alokovaná po častiach o veľkosti PAGE_SIZE, ktorú definuje cieľový počítač a nedá sa nijako užívateľsky ovplyvniť (platforma Intel používa stránky o veľkosti 4 KB, zatiaľ čo na druhých platformách procesorov sa hodnota odlišuje). Existuje niekoľko spôsobov ako alokovať pamäť pre potreby ovládača, podľa toho k čomu je požadovaná. Preto nemusí byť stránkové schéma prideľovania vhodné (požiadavka sa ťažko prispôsobí schéme dvojnásobku veľkosti stránky) a môže viesť k značnému plytvaniu pamäti.

Systém Linux poskytuje alternatívnu funkciu podobnú štandardnej knižničnej malloc s menom kmalloc (kernel memory alloc), ktorá umožňuje prideliť pamäť ľubovoľnej veľkosti v priestore jadra operačného systému. Predávajú sa jej dva parametre: size a flags. Prvým sa udáva množstvo pamäte (počet bajtov), ktorý je vnútorne zaokrúhlený na najbližší násobok veľkosti stránky. Platí však obmedzenie, že je možné prideliť naraz len 128 KB. Pri pokuse alokovať viac pamäti sa objaví správa typu o prekročení veľkosti. Druhým parametrom sa popisuje priorita a atribút stránky pamäti, ktorá sa takto získa. Najčastejšie sa používajú tieto hodnoty:

- GFP_ATOMIC

  Pamäť bude vrátená, ak je nejaká dostupná, bez blokovaného volania alebo získania stránok z odkladajúceho priestoru. Príznak je dobré špecifikovať v prípade pridelenia pamäti pri prerušení, lebo bude zaručené, že aktuálny proces nebude zaradený do fronty, keď nebude príslušná pamäť dostupná.

- GFP_KERNEL

  Pamäť by mala byť vrátená, ak bude nejaká dostupná. Ale v prípade potreby uloženia stránky na disk, môže byť volanie blokované. Je to bežný spôsob nastavenia masky priorít akým sa získava pamäť v jadre.

- GFP_DMA

  Uvedené pre úplnosť. Vrátená pamäť bude alokovaná do hranice 16 MB a tým pádom vhodná pre vyrovnávaciu pamäť DMA. Tento príznak sa využíva len u prenosoch na periférіach zbernice ISA, tie nevedia adresovať viac ako 16 MB pamäti. Zariadenia PCI netrpia týmto obmedzením a pri prenosoch DMA môžu využívať ľubovoľnú pamäť.

V mojom prípade som však použil pri prideľovaní pamäti pre štruktúru mtd_info veľmi výhodnú funkciu kzalloc, ktorá okrem alokácie pamäti túto pamäť na viac aj vynuluje. Príznak je nastavený na štandardný druh pridelenia pamäti, veľkosť štruktúry sa získava makrom sizeof pri predspracovaní.

Navrátenie alokovanej pamäti späť systému je opačná operácia. Je dobré po odobratí modulu ovládača zo systému ale aj pri každej chybe, ktorá sa vyskytla napríklad v priebehu inicializácie, nepotrebnú pamäť okamžite vrátiť. Na uvoľnenie pamäti sa používa funkcia kfree s rovnakým chovaním ako má free funkcia v aplikáciách. Znamená to, že nemení hodnotu svojho parametru a pointer stále ukazuje na miesto, kde bola predtým štruktúra mtd_info. Preto je vhodné pre zabránenie vzniku chýb bezprostredne po uvoľnení daný ukazovateľ nastaviť na NULL.

4.2.5 Registrovanie modulu

Po správnom alokovaní pamäte potrebnej pre mtd_info štruktúry sa pokračuje v naplnení informáciami o modulu ovládača flash pamäti, ktorá ako bude napísané je namapovaný do operačnej pamäti od určitého adresného miesta.

Celková adresovateľná časť na (32 bitových platformách) je rozdelená na dve hlavné časti: priestor pre jadro a užívateľský (alebo aplikačný) priestor. Hranicu definuje makro PAGE_OFFSET, ktoré je možné vidieť a nastavovať v hlavičkovom súbore asm/page.h. Priestor jadra je umiestnený nad ofsetom a užívateľský pod ním. Na platforme Intel ma implicitnú hodnotu 0xC0000000, takže jadro má k dispozícii zhruba 1 GB pamäti a na užívateľský priestor zostávajú 3 GB. Virtuálne adresy sú viditeľné z hľadiska jadra priamo ofsetom fyzickej adresy. Vždy to ale neplatí, a preto musia byť k prevodu využívané príslušné primitíva.

4.2.6 Premapovanie miesta I/O pamäti

Ovládač je abstraktnou vrstvou medzi softwarovým konceptom a hardwarovým obvodom. Každé periférne zariadenie je kontrolované zapisovaním a čítaním z jeho registrov. Sú prístupné buď z adresného priestoru štandardnej pamäti alebo Input/Output. Na hardwarovej úrovni nie je konceptuálny rozdiel medzi pamäťovými regiónmi a I/O regiónmi: oba sú prístupné vystavovaním elektrických signálov na adresu zbernicu a prečítaním alebo zápisom na dátovú zbernicu.

Hlavný mechanizmus spojenia so zariadením sa uskutočňuje pomocou namapovaných registrov do pamäti. I/O pamäť nie je pamäťou v bežnom zmysle (skutočnou RAM pamäťou), ale

skôr vyrovnávajúcou pamäťou na túto oblasť. Kam je namapovaná, závisí predovšetkým na použitej platforme. Funkcia ioremap dovoľuje mapovať fyzické pamäťové miesto I/O pamäti na ukazovateľ jadra o požadovanej veľkosti. Pomocou vráteného ukazovateľa je možné dáta normálne čítať a zapisovať.

Pri uvoľnený modulu z jadra je potrebné premapované oblasti zrušiť. Zabezpečí sa to predaním ukazovateľa funkcii iounmap, ktorá mapovanie odstráni. Vo vytváranom module ovládača som použil túto funkciu pri odregistrovaní modulu z jadra fwh_dev_unreg(), kde sa uvoľňujú všetky alokované štruktúry s ďalším priradením pre úplne ošetrenie možnej chyby (ukazovateľ nastavujem na NULL).

4.2.7 Prístup do I/O pamäti

Na niektorých platformách je možné obísť používanie návratovej hodnoty z ioremap ako ukazovateľ. Takéto použitie nie je prenositeľné medzi rozličnými platformami a správnym postupom získavania obsahu I/O pamäti by malo byť prostredníctvom množiny funkcii zaisťujúcich tento účel definovaných v hlavičkovom súbore asm/io.h.

```
unsigned int ioread8(void *addr);
unsigned int ioread16(void *addr);
unsigned int ioread32(void *addr);
```

Parameter addr by mal byť adresou získanou práve z funkcie ioremap (s prípadným číselným ofsetom). Prečítaná hodnota je tá, ktorú obsahuje I/O pamäť. Taktiež existuje množina funkcii zápisu do I/O pamäti.

```
void iowrite8(u8 value, void *addr);
void iowrite16(u16 value, void *addr);
void iowrite32(u32 value, void *addr);
```

Doteraz popísane funkcie vykonávali operácie nad danou adresou. Na miesto toho pri potrebe pracovať s celým blokom I/O pamäti je možne použiť nasledujúce funkcie so správaním analogickým knižničným funkciám jazyka C.

```
void memset_io(void *addr, u8 value, unsigned int count);
void memcpy_fromio(void *dest, void *source, unsigned int count);
void memcpy_toio(void *dest, void *source, unsigned int count);
```

Pri prehliadaní zdrojových kódov jadra Linuxu je možne naraziť na staršiu množinu funkcii, ktoré stále fungujú, ale sú menej bezpečné, pretože nevykonávajú rovnaký druh typovej kontroly.

4.3 Kompilácia

Jadro je veľký a samostatný program s detailnými a presnými potrebami, ktoré zapadajú do seba. Proces kompilácie a budovania modulov sa významne líši od kompilácií aplikácii vytváraných v užívateľskom režime. Proces pri zostavení jadra Linuxu verzie 2.6 sa taktiež odlišuje od predchádzajúcich verzií. Nový systém je jednoduchší na používanie a produkuje viac správnych výsledkov, ale prináša veci, ktoré sú úplne rozdielne.

Medzi výhody v Linuxe patrí skvelá vlastnosť jadra, že ma schopnosť sa rozširovať počas normálnej prevádzky. To znamená, že je možné pridať nejakú funkcionalitu do jadra (taktiež rovnako odstrániť) pri behu počítača bez nutnosti reštartovať.

Každý kus kódu pridaný do jadra sa nazýva modul. Nový modul sa najprv zo zdrojového textového kódu prekladačom preloží do binárnej formy objektu a potom je spojený, linkovaný s objektom vermagic.o, kedy sa vytvorí špeciálna sekcia v module obsahujúca informácie o verzii jadra, o verzii použitého prekladača, atď. Výsledkom je kompletný zaviteľný modul a môže byť dynamicky naviazaný do jadra. Obecne slovo modul neznamená, že musí byť len zaviteľný za behu operačného systému, ale použité vo význame, že je ho možné zakompilovať aj staticky.

Ešte predtým ako je možné kompilovať moduly jadra je nutné zabezpečiť niektoré predpoklady ako je existencia súčasnej verzie prekladaču, zdrojové kódy celého Linuxu (kernel tree) na disku v adresári /usr/src a iné potrebné knižničné nástroje. Vytvárať jadro a jeho moduly s nesprávnou verziou nástrojov môže viesť k nekonečným, zákerným a spletitým problémom ťažko zvládnuteľných aj pre pokročilých užívateľov. Občas je problematické mať verziu prekladača príliš novú rovnako ako mať starú verziu a taktiež nové vydania (release) zdrojových kódov jadra nemusia mať vždy zaručenú funkčnosť s napísanými ovládačmi.

4.3.1 Vytvorenie súboru pre kompiláciu (Makefile)

Vytvoriť pravidlá pre preloženie zdrojového súboru na objekty zaviteľné do jadra je skutočné priamočiare a jednoduché. Postačuje napísať riadok, ktorý nevyzerá tradične ako pravidlo v bežných dávkových súboroch, pretože zostavovací proces zaviteľných modulov je plne integrovaný do zostavovacieho mechanizmu jadra Linuxu (verzia 2.6), ktoré sa postará o všetko ostatné.

```
obj-m := driver.o
```

Toto priradenie poskytnuté výhodou rozšírenej syntaxe GNU make označuje, že bude vytvorený modul z objektu súboru driver.o (object). Výsledkom po kompilácii objektu súboru je modul jadra s menom driver.ko (kernel object), s novou konvenciou pomenovania predstavenou práve v Linuxe (verzia 2.6) pre odlíšenie od bežných objektov. Ak je potrebné vybudovať modul

jadra z dvoch alebo viacerých zdrojových súborov (napr.: file1.c a file2.c), správnym zápisom je nasledujúce priradenie.

```
NAME = combined
obj-m := $(NAME).o
$(NAME)-objs := file1.o file2.o
```

Sú potrebné dva riadky, kde na spodnom sa uvádza názov binárneho objektu pre zlúčenie zdrojových kódov súborov a ten prvý určuje finálny názov pre výsledný objekt jadra.

Dávkový súbor prekladu musí byť vyvolaný v kontexte zostavovacieho systému jadra. Je nutné presne špecifikovať miesto, kde sa nachádza adresárová štruktúra zdrojových kódov jadra, aby bolo možné zadať príkaz make (program, ktorý spustí dávkový súbor prekladu) v adresári obsahujúcom zdrojový kód modulu (driver.c) a samotný súbor prekladu (Makefile) potrebný na preloženie modulu.

4.3.2 Priebeh prekladu modulu

Preklad zdrojového súboru modulu začína tým, že sa zmení adresár napísaný za voľbou -C, kde sa nachádza dávkový súbor prekladu jadra na najvyššej úrovni a ten riadi ostatok kompilácie. Voľba M spôsobí, že sa vykoná presun späť do adresára so zdrojovým súborom modulu predtým ako sa pokúsi zostaviť cieľ "*modules*". Tento cieľ sa vzápätí odkazuje na zoznam modulov nájdených v obj-m premennej, v ktorej je nastavený názov prekladaného modulu.

Existuje ustálený spôsob využívaný vývojármi, ktorý zaručuje zostavenie modulu mimo štruktúry súborov jadra. Tento malý trik prichádza s možnosťami ponúkanými rozšírenou syntaxou programu GNU make. Dávkový súbor prekladu (Makefile) je potom prečítaný a prechádzaný dvakrát pri typickom zostavovaní.

Keď je vyvolaný z príkazového riadka, začína prvý priebeh. Do pozornosti sa dostáva premenná KERNELRELEASE, ktorá nie je nastavená.

```
ifneq ($(KERNELRELEASE),)
```

Preklad pokračuje vetvou, kde sa nastaví cesta so zdrojovými textami do premennej PWD (aktuálna pozícia adresára), ale hlavne názov umiestnenia adresára zdrojových súborov jadra do premennej KERNELDIR. Využíva sa skutočnosť, že existuje vytvorený symbolický odkaz build späť na začiatok stromovej štruktúry jadra. Ten je umiestnený v adresári knižníc modulov a mal by byť definovaný pre každú verziu Linuxu.

```
PWD := $(shell pwd)
KERNELDIR ?= /lib/modules/$(shell uname -r)/build
```

Ak aktuálne nebeží verzia jadra Linuxu, pre ktorý sa zostavuje modul, je možné dodať voľbu pre nastavenie premennej KERNELDIR priamo v príkazovom riadku, alebo nastaviť premennú prostredia, alebo prepísať riadok v dávkovom súbore na konkrétny požadovaný názov adresára zdrojových súborov jadra. Ak je správne nájdený tento adresár, dávkový súbor prekladu vyvolá implicitný default cieľ, ktorý spustí druhý priebeh kompilácie. Príkaz je parametrizovaný so zistenými voľbami a spustí zostavovací systém jadra.

```
default:
    $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
```

Pri druhom čítaní dávkového súboru je nastavená premenná obj-m a o zvyšok kompilácie sa už postará dávkový súbor jadra jeho zostavovací systém presne tak ako bolo napísané predtým.

Celý tento mechanizmus pre kompiláciu modulu môže pripadať trochu nemotorný a nezrozumiteľný, ale výhodou sú schopnosti, ktoré boli naprogramované v zostavovacom systéme jadra a je praktické ich používať. Reálny kompletný súbor prekladu by mal zahrňovať ešte aj ciele pre vyčistenie a vymazanie nepotrebných súborov, prípadne inštalácie modulu do systému a pod. Príklad korektného výstupu na konzolu pri kompilácií je zachytený v prílohe.

5 Mechanizmus upgradu

Výsledkom mojej práce je vytvorenie aplikácie (daemon), ktorá bude ponúkať rozhranie medzi vytvoreným ovládačom a knižnicou OpenIPMI. Podrobnosti o nej sú rozobrané v 6. kapitole. Napojenie na ovládač je skutočne jednoduché. Ako už bolo napísane, pristupovať k zariadeniam je v Linuxe uskutočnené pomocou súborového systému, takže je potrebné otvoriť a pracovať so správnym súborom. Tieto informácie poskytuje virtuálny súborový systém (Procfs).

Pomocou neho môže používateľ nahliadnuť do vnútorných premenných jadra alebo ovládačov. Primárne bol vyvinutý na správu procesov, ale Linux z neho urobil veľmi použiteľný nástroj na správu celého systému. Virtuálny znamená, že žiaden súbor sa nenachádza na disku, ale všetko je v pamäti, aj keď vstupom je adresár /proc. Zaberá len málo operačnej pamäte a je veľmi dynamický. Obsah súborov je textový a tým sa jednoducho dokážu prečítať rôzne podrobnosti o niektorých druhoch hardvéru. MTD poskytuje informácie o aktuálnom stave natiahnutých modulov, ich pomenovanie, rozsah veľkosti a iné.

Nasledujúci výpis prevzatý z konzoly to zobrazuje:

root@test:/home/igor/fwh_dev# **cat /proc/mtd**

dev:	size	erasesize	name
mtd0:	00080000	00001000	"MEMORY"
mtd1:	00080000	00001000	"REGISTER"

Použitie celého programu (daemon) spočíva v schopností prepísať obsah flash pamäte BIOSu kopírovaním pripraveného binárneho súboru do oblasti *Memory*. Pred samotným zápisom musia byť však povolené všetky ochrany zápisu. Tieto povolenia sa vykonajú zapísaním definovaných hodnôt na konkrétne miesta v oblasti *Register* (viz. tabuľka 7). Aby to bolo možné je potrebné povoliť na kontrolére matičnej dosky transformáciu a mapovanie adries do operačnej pamäte pre daný flash čip. Mechanizmom majú byť tieto povolenia zápisu realizované čo najjednoduchšie.

Názov bloku	**Číslo bloku**	**Chránený rozsah adries v oblasti Memory**	**Mapovaná adresa Register**
T_BLOCK_LK	Block 7	07FFFFH – 070000H	FFBF 0002H
T_MINUS01_LK	Block 6	06FFFFH – 060000H	FFBE 0002H
T_MINUS02_LK	Block 5	05FFFFH – 050000H	FFBD 0002H
...	...	...	...
T_MINUS07_LK	Block 0	00FFFFH – 000000H	FFB8 0002H

Tabuľka 7: Adresy Block Locking registrov pre povolenie zápisu konkrétnych blokov.

5.1 Definovanie rozhrania aplikácie

Vytvorená aplikácia predstavuje bežný program so zadefinovanou voľbou možností, navrhnutá tak aby bolo možné ďalšie voľby jednoducho pridávať. Vstupným parametrom je prepínač, na základe ktorého sa vykoná požadovaná operácia. Jeho možností sú: čítanie, zápis a vymazanie obsahu flash pamäti čipu. Druhým požadovaním vstupným parametrom je názov binárneho súboru použitý pre upgrade BIOSu.

Vykonanie povolenia zápisu predchádza samotnej aktualizácii a načítava z konfiguračného súboru údaje o miestach registrov a nastavovaných hodnôt. Takže je potrebné pred operáciou povoliť, prípadne zamedziť prístup po ukončení, použitím prepínača a názvu tohto súboru.

5.2 Zbernica PCI

PCI (Peripheral Component Interconnect) je počítačová zbernica pre pripojovanie vnútorných komponentov dohromady v systéme. Štandard popisuje spôsob, akým sú komponenty elektricky prepojené a ich správanie. Dôležité sú parametre zbernice: dátová šírka (32 alebo 64 bitov), adresný priestor pamäti a portov, geografická adresa zariadenia (identifikovaná slotom), a iné technické parametre. PCI nie je len doménou platformy x86, ale používa sa aj na systémoch Alpha, SPARC64, PowerPC a ďalšie.

Zbernica PCI predstavuje okrem spôsobu rozloženia elektrických drôtov v skutočnosti doplňuje špecifikáciu definovaním ako sa rozličné časti počítača vzájomne ovplyvňujú. Cieľom PCI architektúry bolo zostrojiť náhradu pre zastaranú zbernicu ISA s troma hlavnými požiadavkami: získať lepši výkon pri prenose dát medzi počítačom a perifériami, urobiť ju platformovo nezávislú a jednoduchým spôsobom pridávať a odberať komponenty v systéme.

5.2.1 Adresovanie

Každé PCI zariadenie je identifikované číslom zbernice (bus number), číslom zariadenia (device number) a číslom funkcie (function number). Špecifikácia povoľuje mať v obyčajnom systéme dostupné až 256 zberníc, a však toto číslo nie je dostatočne veľké pre počítačové systémy, preto Linux podporuje PCI domény (domains). Každá doména môže potom hosťovať až 256 zberníc, čím sa stáva celý systém uspokojujúci pre veľké množstvo zariadení. Každá zbernica zahŕňa 32 zariadení a každé zariadenie môže byť multifunkčné spravované s maximálnym počtom 8 funkcií. Preto každá funkcia je identifikovateľná na hardwarovej vrstve 16 bitovou adresou.

V moderných pracovných staniciach vystupujú prinajmenšom dve zbernice. Zapojením viac ako jednej zbernice v jednom systéme je dosiahnuté prostriedkami nazývanými most (bridge). Je to špeciálne určený komponent, ktorého úlohou je spájanie dvoch zberníc. Celkový globálny návrh PCI systému tak vytvára stromovú štruktúru, kde každá zbernica je napojená na vrchnú vrstvu až

k zbernici nula, koreňu celého stromu. Podobne pomocou LPC mostu je prepojená LPC zbernica, na ktorú je pripojený flash pamäť čip BIOSu (viz. Obrázok 4.1).

Každé zariadenie na doske musí byť najprv geograficky adresovateľné, aby sa získali jeho konfiguračné registre, s ktorými je potom možne vykonávať normálne vstupne/výstupné operácie ako pri prístupe do systémovej pamäti. Konfiguračný priestor pozostáva 256 bajtov pre každú funkciu zariadenia (okrem PCI Express zariadenia ktoré má tento priestor veľký 4 KB) a usporiadanej štruktúry registrov, ktoré sú štandardizované. Štyri bajty konfiguračného priestoru udržujú unikátne ID ustanovené pre všetky zariadenia. Ostávajúci priestor je použitý pre nastavenie správania samotného zariadenia, ktorý sám definuje aké hodnoty na akých ofsetoch je možné použiť.

5.2.2 Príklad registra PCI

Adresa ofsetu:	0088H
Základná Hodnota:	0100 00F0H
Atribút:	R/W
Veľkosť	32 bitov

Premapovaná pamäť BIOSu môže byť chránená čipsetom matičnej dosky, aby sa predchádzalo nepovolenému čítaniu alebo zápisu do tohto regiónu. 20 bitový región môže byť tiež premapovaný na vrchol systémovej pamäti pre zaručenie kompatibility. Všetky nastavované kontrólne bity môžu byť uzamknuté pre bezpečnosť.

Bity:	Označenie:	Popis:
31-25	rezervované	
24	SSP_XLAT	Programová ochrana zbernicového cyklu prekladu adresy. 0 = vypne všetky prevody adries.
23	rezervované	
22-21	BOOT_BIOS_DEC	Kontroluje cieľ cyklu. 00 = Rozsah BIOSu je dekódovaný ISA mostom a presmerovaný na LPC zbernicu.
20	LK_LOCK	Kontroluje prístup do združených registrov CTL1, CTL2, CTL3 a DEC. 1 = uzamknutý, zapisovanie nie je možné.
19-8	rezervované	
7-0	ROM_SEGMENT	Kontroluje zodpovedajúci segment pamäti, ktorý je dekódovaný ako BIOS = 1 alebo ako normálna systémová pamäť = 0.

5.3 Konfiguračný súbor

Obsahuje nastavenia hodnôt povolenia zápisu pre upgrade BIOSu do spomínanej oblasti pamäti. Je to textový súbor, aby bol prenositeľný a jednoducho upravovateľný v každej situácii. Zvyčajne všetky nastavenia majú tvar "kľúč" = "hodnota" a vytvárať niečo, čo je už odladené na takéto použitie je zbytočné.

Na jeho spracovanie a získanie hodnôt som využil už hotovú knižnicu pre parsrovanie súborov. Touto pomocnou knižnicou je libConfuse napísaná tiež v jazyku C, ktorá poskytuje okrem základnej požiadavky aj podporu pre sekcie (zoznamy) rozličných hodnôt (reťazcov, celých i reálnych čísel, logických údajov a vnorených sekcií). Vytvára tak spôsob, ako nenáročne pridávať, spracovávať a robiť rozbor konfiguračného súboru používaním jej zrozumiteľného rozhrania ľahko integrovateľného do kódu programu.

Na oficiálnych stránkach sú k dispozícii zdrojové kódy knižnice, ako aj manuál (odkaz na dokumentáciu) a vysvetľujúca nápoveda (tutorial) k použitiu. Stiahnuté kódy je nutné preložiť a nainštalovať. Výsledkom je statická knižnica umiestnená v adresári /usr/local/lib.

Štruktúra konfiguračného súboru zohľadňuje formulované východiská. Pre každý register v PCI konfiguračnom priestore je nutné definovať, ku ktorému zariadeniu sa pristupuje, čo je udáva trojica zbernica, zariadenie, funkcia (bus, device, function). Ďalej sa určí ofset (offset) a zapisovaná hexadecimálna hodnota (value) v rozsahu jednotiek (unit) 8, 16 alebo 32 bitov (byte, word alebo doubleword). Nasledujúci príklad demonštruje jeden prvok zoznamu nastavení:

```
WriteEnablePCI "regDEC"
{
  PCIdev = {0, 1, 0}
  Offset = 0x88
  Value = 0x010000f0
  Unit = "dw"
}
```

Podobne je formovaný aj prístup do registrov flash pamäti oblasti Memory. Každá položka predstavuje adresu (address), na ktorú sa zapíše hodnota (value) danej veľkosti (unit).

```
WriteEnableMEM "BLK07"
{
  Addr = 0xffbf0002
  Value = 0x00
  Unit = "b"
}
```

6 Knižnica OpenIMPI

IPMI je skratka Intelligent Platform Management Interface. Základným porozumením IPMI je vyžadované pri používaní knižnice OpenIPMI, ktorá skrýva detaily zasielania správ a dátových formátov príkazov.

Štandard IPMI poskytuje možnosti vzdialeného prístupu a administráciu serverov. Nová verzia špecifikácie 2.0 začína byť teraz podporovaná mnohými servermi do rackových skrini, bola prijatá viac než 150 výrobcami serverových technológii a aj napriek tomu nie je stále veľmi rozšírená v podnikových sieťach.

Je vyžadované mať vedľa hlavného procesoru ďalší fungujúci špeciálny procesor nazývaný BMC (Baseboard Management Controler), ktorý je vždy zapnutý a udržuje kontrolu nad systémom. Vo väčšine systémoch s IPMI je možne monitorovať a spravovať hardwarové súčasti, dokonca keď je hlavný procesor vypnutý, a však celý systém musí byť zapojený v elektrickej sieti. IPMI sa zaoberá platformou a nie softwarom bežiacim na nej. Spravovanie softwaru je väčšinou mimo oblasť pôsobenia.

Riadenie (správa) systému znamená odlišné veci pre rozličné odvetvia. V obyčajných serverových systémoch ma úlohu kontrolovať zapnutie a prívod elektriny, zabezpečiť stálu teplotu. V telekomunikačných systémoch znamená riadenie, kontrolu nad všetkými stránkami systému od naštartovania všetkých súčasti, plné monitorovanie komponentov, detekciu a zotavenie z chýb. V minulosti bolo takéto kontrolovanie a správa systému realizované pomocou množstva patentovaných rozhraní, ktoré IPMI štandardizuje.

6.1 Štruktúra príkazov IPMI

Procesor BMC je spojený s hlavným procesorom CPU (Central Processor Unit) a ďalšími jednotkami matičnej dosky (motherboard) jednoduchou sériovou zbernicou. Pomocou nej monitoruje základné parametre: teplotné senzory, status CPU, otáčky ventilátorov a iné.

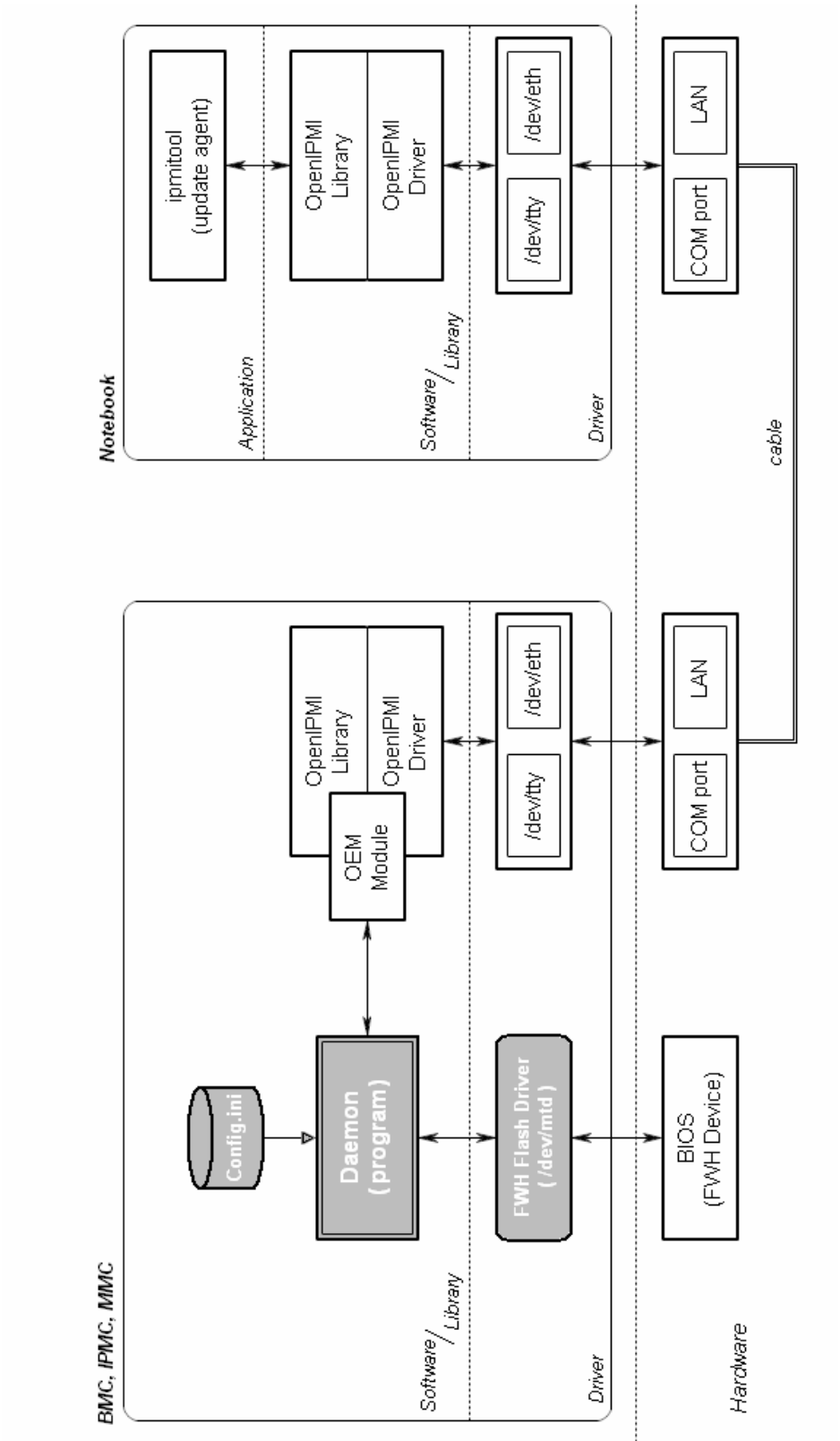
Prístup k informáciám poskytovaným procesorom BMC je možný pomocou sériovej konzoly tak aj rozhraním protokolu IP užitím terminálu (napr.: PuTTY, TeraTerm). Komunikácia prebieha pomocou zápisu požiadaviek, na ktoré prichádzajú odpovede v definovanom tvare. Všetky požiadavky sú v skutočnosti pole rôznej dĺžky (podľa príkazu) hexadecimálnych celých čísel oddelených medzerami vymedzených v hranatých zátvorkách. Nasledujúci príklad ukazuje zaslanie požiadavku pre zistenie ID zariadenia (Get Device ID) zapísaný v konzole terminálu.

```
[18 xx 01]
```

Prvý bajt – číslo predstavuje Network Function Code, deklarácia množiny príkazov. Druhý je Request Sequence Number, poradové číslo vyslaného príkazu a tretí Command Code, čiže presná špecifikácia príkazu.

Vo väčšine existujú dlhšie príkazy s počtom prekračujúcich 3 hexadecimálne čísla, ale práve tieto tri sa musia vyskytovať vždy v každom z nich presne v poradí podľa štandardu špecifikácie. Zrejme je jasné, že pri takomto zasielaní a získavaní správ je efektivita riadenia systému značne spomalená, preto je potreba, aby bola vhodne zapuzdrená. Uľahčením a nevyhnutnosťou pre komunikáciu je dobré použiť utilitu kompatibilnú so štandardom IPMI ako je Linuxová aplikácia ipmitool. Tento nástroj je aplikáciou využívajúcou knižnicu OpenIPMI.

6.2 Jednoduchý návrh



Obrázok 6.1: Schéma komunikácie systému riadenia s vytvorenou aplikáciou.

7 Záver

Množstvo počítačových systémov vyžaduje pri svojom spúšťaní BIOS - Basic Input Output System pre zaistenie, že všetky komponenty fungujú správne, aby bol nakoniec zavedený operačný systém. Preto je neoddeliteľnou súčasťou počítačov a je potreba jeho aktualizácie pri zmene hardwarovej konfigurácie alebo výskyte problému.

Úlohou bolo naštudovať, pripraviť technické podklady pre zvládnutie implementácie aktualizácie BIOSu v prostredí operačného systému Linux (verzia 2.6). Pri vytáraní ovládača pre zariadenie, písania zdrojových kódov modulu jadra, je dobrým predpokladom mať primerané znalosti programovacieho jazyka C. Výsledkom práce je vytvorený ovládač pre konkrétny typ flash pamäti, v ktorej je uložený obsah BIOSu, využitím rozhrania subsystému MTD. Je pripravený s dávkovým súborom ku kompilácií k použitiu.

Pri riešení som vychádzal z možností, ktoré poskytuje tento subsystém: možnosť jednoducho pristúpiť k zariadeniu a prečítať jeho obsah, ďalej prepísať obsah novými dátami, ktorému predchádza prvotné vymazanie blokov. Veľmi dôležitú rolu zohráva skutočnosť, že každá flash pamäť má presne definovanú softwarovú sekvenciu príkazu. Vyžadovaný cyklus stanovuje najprv zápis definovaných hodnôt na určité adresy, kedy sa uvedie vnútorná logika čipu do zapisovacieho príp. mazacieho stavu. To je riešené práve na úrovni ovládača, ktorý je rozhraním medzi fyzickou vrstvou a operačným systémom resp. užívateľskými aplikáciami.

Táto nevedomosť mi spôsobila problém so zápisom žiadanej hodnoty do premapovanej Input/Output pamäti, keď čítanie prebiehalo úplne v poriadku. Však zápis ani len jedného bajtu do oblasti Register vôbec nefungoval, čo spôsobilo zdržanie testovania funkčnosti ovládača a následne užívateľskej aplikácie celého mechanizmu.

Literatúra

[1] Corbet J., Rubini A., Kroah-Hartman G.: Linux Device Drivers, Third Edition. O'Reilly Media, 2005, ISBN: 0-596-00590-3

[2] Matthew N., Stones R.: Linux Začíname programovat, Kapitola 3: Práce se soubory, Kapitola 21: Ovladače zařízení. Computer Press, 2000, ISBN: 80-7226-307-2

[3] Coffey T., O'Shaughnessy A.: Write a Linux Hardware Device Driver, CMP Media LLC, [máj 2008], Článok dostupný na URL: http://www.networkcomputing.com/unixworld/tutorial/010/010.txt.html

[4] The Linux MTD project, oficiálne stránky projektu, [máj 2008], Dostupné na URL: http://www.linux-mtd.infradead.org/

[5] MTD internal API documentation, dokumentácia k MTD, [máj 2008], Dostupná na URL: http://www.linux-mtd.infradead.org/archive/tech/

[6] Corbet J.: *Porting device drivers to the 2.6 kernel*, cenné zdroje informácií o tvorbe ovládačov, [máj 2008], Dostupné na URL: http://lwn.net/Articles/driver-porting/

[7] SST49LF040B 4 Mbit LPC Flash (datasheet product), [máj 2008], Dostupné na URL: http://www.sst.com/products.xhtml/serial_flash/49/SST49LF040B/

Zoznam príloh

Príloha 1.

Výpisy z konzoly Linuxu po zadaní niektorých príkazov

```
root@test:/home/igor/fwh_dev# make
make -C /lib/modules/2.6.22-14-generic/build M=/home/igor/fwh_dev modules
make[1]: Entering directory `/usr/src/linux-headers-2.6.22-14-generic'
  CC [M]  /home/igor/fwh_dev/fwh_dev.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/igor/fwh_dev/fwh_dev.mod.o
  LD [M]  /home/igor/fwh_dev/fwh_dev.ko
make[1]: Leaving directory `/usr/src/linux-headers-2.6.22-14-generic'


root@test:/home/igor/fwh_dev# modinfo ./fwh_dev.ko
filename:       ./fwh_dev.ko
version:        develop ver. 1.1
description:    MTD flash driver for specific FirmWare Hub device
author:         Igor Mariscak <igormariscak@hotmail.com>
license:        GPL
srcversion:     E5BC5E3199395E466E55602
depends:        mtd
vermagic:       2.6.22-14-generic SMP mod_unload 586


root@test:/home/igor/fwh_dev# modinfo /lib/modules/2.6.22-14-
generic/kernel/drivers/mtd/mtd.ko
filename:       /lib/modules/2.6.22-14-generic/kernel/drivers/mtd/mtd.ko
description:    Core MTD registration and access routines
author:         David Woodhouse <dwmw2@infradead.org>
license:        GPL
license:        GPL
srcversion:     6179BF6537506E1C8F511A3
depends:
vermagic:       2.6.22-14-generic SMP mod_unload 586


root@test:/home/igor/fwh_dev# insmod ./fwh_dev.ko
root@test:/home/igor/fwh_dev# dmesg | tail -n3
[ 2679.212000] fwh_dev: Register the region part 'MEMORY' with MTD subsystem
[ 2679.216000] fwh_dev: Register the region part 'REGISTER' with MTD subsystem
[ 2679.216000] fwh_dev: Insert module of FWH device 'SST49LF040B' successful
```

```
root@test:/home/igor/fwh_dev# lsmod
Module                  Size  Used by
...
fwh_dev                 7300  0
mtdchar                 9348  0
mtd                    17412  5 fwh_dev,mtdchar
...


root@test:/home/igor/fwh_dev# cat /proc/modules
...
fwh_dev 7300 0 - Live 0xf8c93000
mtdchar 9348 0 - Live 0xf8cc7000
mtd 17412 5 fwh_dev,mtdchar, Live 0xf8cd9000
...


root@test:/home/igor/fwh_dev# rmmod ./fwh_dev.ko
root@test:/home/igor/fwh_dev# dmesg | tail -n3
[ 4077.572000] fwh_dev: Un-Register the region part 'MEMORY' from MTD subsystem
[ 4077.572000] fwh_dev: Un-Register the region part 'REGISTER' from MTD subsystem
[ 4077.572000] fwh_dev: Remove module of FWH device 'SST49LF040B' successful
```

Príloha 2.

Stručná špecifikácia LPC zbernice

Veľký počet pinov na zariadeniach potrebných pre podporu zberníc ISA (Industry Standard Architecture) a iných štandardov zvyčajne zvyšovali celkovú cenu systému. Vyžadované rozmerné pätice na usadenie CPU ale najmä mikrokontrolérov na základnej doske stále nútili výrobcov vymýšľať nové proprietárne implementácie zberníc. Vývoj LPC (Low Pin Count) zbernice mal predísť a vyhnúť sa niektorým spomínaným problémom.

Zbernica LPC realizuje obecný cieľ signálov na linke, ktoré v podstate prenášajú časovo multiplexované adresy, dáta a kontrolné informácie za účelom vykonať pamäťové, vstupne/výstupne (V/V) a zbernicové transakcie medzi procesorom a inými systémovými zariadeniami.

Špecifikácia definuje 7 dôležitých a povinných signálov potrebných pre obojsmerný prenos: len 4 na multiplexovanie príkazu, adresy a dát; zostávajúce 3 sú kontrolné signály (frame, reset a clock). Ďalej definuje voliteľných 6 signálov, ktoré môžu byť použité pre podporu prerušenia (interrupt), priamy prístup do pamäti (DMA), zobudiť systém z módu nízkej spotreby, oznámenie komponentom o nasledujúcom odpojení elektrickej energie a pod. Prenosová rýchlosť V/V operácií pri taktovaní 33,3MHz je 132Mbit/s.

Je navrhnutá ako lokálna zbernica obsluhujúca obvodové komponenty na základnej doske. Doposiaľ neexistuje žiadny periférny prvok, ktorý by bol k nej pripojený cez konektor, a rovnako neposkytuje ani rozšíriteľnosť pre doplňujúce vlastnosti, čo je zaistené štandardom PCI (Peripheral Component Interconnect) zbernicou. Vo všeobecnosti môže byť LPC limitujúca pre vzájomné spojenie v počítačovom systéme definujúca styčné rozhranie kontroléru a jedného či viacerých pamäťových zariadení. Potom používaný termín "***Firmware Hub***" sa vzťahuje na pamäťové zariadenia prepojené so zbernicou LPC.

Technická špecifikácia rozhrania je dostupná na stránkach spoločnosti Intel[V], stiahnutá na priloženom kompaktnom disku.

Vlastnosti použitej flash pamäti

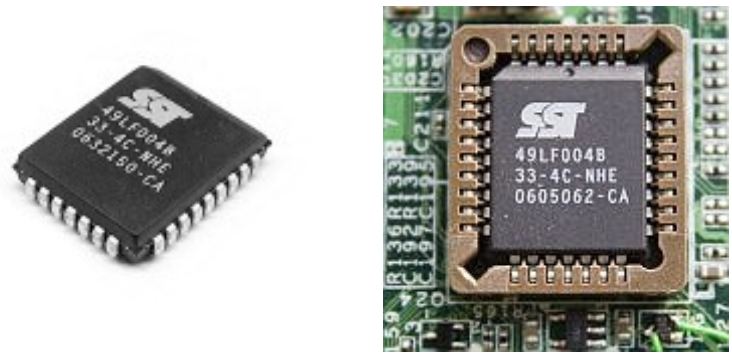
Rozdiely medzi SST49LF040B a SST49LF004B sú v internom zapojení a spôsobe komunikácie: prvý typ odpovedá na LPC Memory typ cyklu, kým druhý na Firmware Memory cyklus a je spätne kompatibilný so skoršími revíziami.

Tento produkt flash pamäť firmy SST je vyrábaný s patentovanou vysoko-výkonnou SuperFlash technológiou. Bunky zostrojené oddelenými hradlami pracujúce na princípe Tunelového efektu. Je to jav v kvantovej mechanike, kedy častice môžu preniknúť cez oblasť, na prekonanie

[V] Špecifikácia dostupná: http://www.intel.com/design/chipsets/industry/lpc.htm

ktorej by podľa klasickej fyziky nemali dostatočnú energiu[VI]. Takto dosahujú väčšiu spoľahlivosť v porovnaní s alternatívnymi výrobnými prístupmi. Pri významnom zdokonalení spoľahlivosti a výkonu flash pamäti spotreba napájania energie sa však znižuje. Použije sa celkovo menej elektrického prúdu počas mazacieho alebo programovacieho cyklu pamäťových operácií ako je to pri podobných pamäťových technológiách. Poskytuje taktiež fixné časy trvania operácií, nezávislé na počte cyklov, ktoré sa už odohrali. To znamená, že nie je vôbec potrebné hardvérovo či programovo kalibrovať dĺžky časov s narastajúcim počtom mazacích alebo programovacích cyklov, čo je uľahčením a výhodou.

Poskytuje operáciu zápisu jedného bajtu s maximálnym programovým časom 20 µsek, celú pamäť je možné preprogramovať typicky za 8 sek. Z ďalších ponúkaných znakov je prívetivá odolnosť na opakovaný úspešný zápis, ktorú zariadenie ešte znesie, približne 100 000 cyklov. Doba uchovania zapísaných dát sa odhaduje na viac ako 100 rokov. Čip sa dodáva v dvoch rozličných fyzických baleniach (package) bez škodlivých environmentálnych látok (bez olova). Kompletný prehľad fyzikálnych vlastností je uložený na CD vo forme podrobného záznamového listu (datasheet) získaného z oficiálnych stránok[VII].



Reálne zobrazenie flash pamäti čipu pre uloženie BIOSu.

[VI] Podrobné vysvetlenie: http://en.wikipedia.org/wiki/Quantum_tunnelling

[VII] Silicon Storage Technology: http://www.sst.com/